

# Học Angular từ cơ bản

 chatgpt.com/c/67d6cc18-dca8-8001-85ba-3c13abc0c1b6

# ChatGPT

Hướng dẫn tôi học angular từ cơ bản đến nâng cao

## Lộ trình học Angular từ cơ bản đến nâng cao

Để học **Angular** hiệu quả, bạn nên chia lộ trình thành từng giai đoạn với các kiến thức từ cơ bản đến nâng cao. Dưới đây là hướng dẫn chi tiết giúp bạn học Angular một cách có hệ thống.

### ◆ 1. Kiến thức nền tảng trước khi học Angular

Trước khi bắt đầu với Angular, bạn cần nắm vững một số kiến thức sau:

- **HTML, CSS, JavaScript (ES6+)**: Biết về DOM, sự kiện, bất đồng bộ (Promises, Async/Await), ES6+ (arrow functions, destructuring, modules).
- **TypeScript**: Angular sử dụng TypeScript, vì vậy bạn nên học về:
  - Kiểu dữ liệu (`number`, `string`, `boolean`, `array`, `object`, `any`, `void`, `union types`, `interface`)
  - Lập trình hướng đối tượng trong TypeScript (OOP: Class, Interface, Inheritance)
  - Generic, Decorators, Modules

### ◆ 2. Học Angular cơ bản

## 2.1. Cài đặt môi trường

---

- Cài đặt **Node.js** (phiên bản mới nhất)
- Cài **Angular CLI** bằng lệnh:

```
bash
```

```
npm install -g @angular/cli
```

- Tạo dự án Angular:

```
bash
```

```
ng new my-angular-app  
cd my-angular-app  
ng serve
```

## 2.2. Cấu trúc dự án Angular

---

Tìm hiểu về cấu trúc của một ứng dụng Angular:

- **src/app/** - Thư mục chứa code chính.
- **src/assets/** - Nơi chứa ảnh, file tĩnh.
- **src/environments/** - Chứa file cấu hình môi trường (dev, production).
- **angular.json** - Cấu hình Angular CLI.
- **package.json** - Quản lý dependencies của dự án.

## 2.3. Components - Thành phần chính của Angular

---

- Tạo một Component:

```
bash
```

```
ng generate component my-component
```

- Các thành phần chính:
  - **Template (HTML)**: Giao diện của component.
  - **Style (CSS, SCSS)**: Định dạng giao diện.
  - **Class (TypeScript)**: Chứa logic xử lý.
  - **Metadata (Decorator @Component)**: Cấu hình component.

Ví dụ một component:

```
typescript
```

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

@Component({
  selector: 'app-hello',
  template: `<h1>{{ title }}</h1>`,
  styleUrls: ['./hello.component.css']
})
export class HelloComponent {
  title = 'Chào mừng đến với Angular!';
}
```

## 2.4. Data Binding - Liên kết dữ liệu

---

- **Interpolation ({{}}):**

```
html
```

```
<h1>{{ title }}</h1>
```

- **Property Binding ([ ]):**

```
html
```

```
<input [value]="name">
```

- **Event Binding (( )):**

```
html
```

```
<button (click)="sayHello()">Click me</button>
```

- Two-way Binding (**[()]**):

```
html
```

```
<input [(ngModel)]="name">
```

## 2.5. Directive - Chỉ thị

---

- Structural Directive (**\*ngIf**, **\*ngFor**):

```
html
```

```
<p *ngIf="isLoggedIn">Welcome!</p>
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

- Attribute Directive (**[ngClass]**, **[ngStyle]**):

```
html
```

```
<p [ngClass]="{ 'active': isActive }">Class Binding</p>
```

## 2.6. Services và Dependency Injection

---

- **Tạo Service:**

```
bash
```

```
ng generate service my-service
```

- **Sử dụng Service:**

```
typescript
```

```
@Injectable({
  providedIn: 'root',
})
export class MyService {
  getData() {
    return ['Item 1', 'Item 2'];
  }
}
```

---

## ◆ 3. Học Angular nâng cao

---

### 3.1. Routing - Điều hướng

---

- **Cấu hình Routing:**

typescript

```
const routes: Routes = [  
  { path: 'home', component: HomeComponent },  
  { path: 'about', component: AboutComponent },  
  { path: '**', redirectTo: 'home' }  
];
```

- **Điều hướng bằng code:**

typescript

```
this.router.navigate(['/about']);
```

## 3.2. Reactive Forms

---

- **Form Control và Validation:**

typescript

```
this.loginForm = new FormGroup({  
  username: new FormControl('', Validators.required),  
  password: new FormControl('', [Validators.required,  
    Validators.minLength(6)])  
});
```

- **Sử dụng trong Template:**

html

```
<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
  <input formControlName="username">
  <input formControlName="password" type="password">
  <button type="submit">Login</button>
</form>
```

### 3.3. HTTP Client - Gọi API

---

- **Cài đặt HttpClient:**

typescript

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

constructor(private http: HttpClient) {}

getUsers() {
  return this.http.get('https://jsonplaceholder.typicode.com/users');
}
```



- Sử dụng trong Component:

```
typescript
```

```
this.userService getUsers().subscribe(users => {  
  this.users = users;  
});
```

### 3.4. State Management với NgRx

---

- Cài đặt NgRx:

```
bash
```

```
npm install @ngrx/store @ngrx/effects
```

- **Tạo Store và Reducer:**

typescript

```
export interface AppState {  
  counter: number;  
}  
  
export const initialState: AppState = { counter: 0 };  
  
export function counterReducer(state = initialState, action) {  
  switch (action.type) {  
    case 'INCREMENT':  
      return { counter: state.counter + 1 };  
    default:  
      return state;  
  }  
}
```

---

## ◆ 4. Học Angular chuyên sâu

- **Lazy Loading Modules:** Chia nhỏ ứng dụng thành các module con để tối ưu hiệu suất.
- **SSR với Angular Universal:** Render phía server để cải thiện SEO.
- **Testing trong Angular:** Viết Unit Test với Jasmine/Karma.
- **Bảo mật trong Angular:** Sử dụng AuthGuard, JWT Authentication.

---

## 5. Tài nguyên học Angular

- **Trang chủ Angular:** <https://angular.io/>
- **Angular University:** <https://angular-university.io/>
- **Khóa học Angular trên Udemy:**
  - *Angular - The Complete Guide (Maximilian Schwarzmüller)*
  - *Angular & NgRx - Build an App with Angular and Redux*
- **Khóa học miễn phí trên YouTube:**  
*Codevolution, Academind, Traversy Media*

---

## Lời khuyên

- Hãy thực hành nhiều, xây dựng các dự án nhỏ để hiểu rõ cách hoạt động của Angular.
- Đọc tài liệu chính thức để cập nhật kiến thức mới nhất.
- Khi gặp lỗi, hãy tra cứu trên Stack Overflow hoặc GitHub Discussions.

Bạn đã có nền tảng về React.js nên có thể tiếp cận Angular nhanh hơn bằng cách so sánh các khái niệm tương tự như Component, State Management, Routing.

Bạn có cần hỗ trợ gì thêm không? 🚀

Bây giờ hãy bắt đầu dạy tôi theo thứ tự cơ bản đến nâng cao

## Bài 1: Giới thiệu Angular và Cài đặt Môi trường

---

### 1. Angular là gì?

---

**Angular** là một framework frontend phát triển bởi Google để xây dựng ứng dụng web động, hiệu suất cao. Nó sử dụng **TypeScript** thay vì JavaScript thuần, hỗ trợ lập trình hướng đối tượng và đi kèm với nhiều tính năng mạnh mẽ như **Component-Based Architecture**, **Dependency Injection (DI)**, **Routing**, và **State Management**.

---

### 2. Cài đặt Môi trường

---

Trước khi lập trình với Angular, bạn cần chuẩn bị các công cụ sau:

#### 2.1. Cài đặt Node.js

---

Angular yêu cầu **Node.js** để quản lý các package thông qua **npm**.

👉 Tải và cài đặt **Node.js** tại: <https://nodejs.org/>

Kiểm tra phiên bản Node.js và npm:

```
bash
```

```
node -v  
npm -v
```

#### 2.2. Cài đặt Angular CLI

---

**Angular CLI** là công cụ giúp bạn tạo, xây dựng và quản lý ứng dụng Angular. Cài đặt bằng lệnh:

```
bash
```

```
npm install -g @angular/cli
```

Kiểm tra phiên bản Angular CLI:

```
bash
```

```
ng version
```

## 2.3. Tạo Dự Án Angular

---

Sau khi cài đặt xong **Angular CLI**, bạn có thể tạo một dự án mới:

```
bash
```

```
ng new my-angular-app
```

**Tùy chọn:** Nếu CLI hỏi về cấu hình, bạn chọn như sau:

- **Would you like to add Angular routing?** → Chọn **Yes**
- **Which style sheet format would you like to use?** → Chọn **SCSS** hoặc **CSS**

Di chuyển vào thư mục dự án:

```
bash
```

```
cd my-angular-app
```

## 2.4. Chạy Ứng Dụng Angular

---

Dùng lệnh sau để chạy ứng dụng:

```
bash
```

```
ng serve
```

Mở trình duyệt và truy cập: **<http://localhost:4200>**

Bạn sẽ thấy giao diện mặc định của Angular.






---


## 3. Cấu trúc dự án Angular

---

Khi tạo một dự án Angular, bạn sẽ thấy các thư mục sau:

 **src/**

-  **app/** → Chứa toàn bộ mã nguồn của ứng dụng.
-  **assets/** → Chứa hình ảnh, file tĩnh.
-  **environments/** → Chứa cấu hình môi trường (dev, prod).
-  **main.ts** → Điểm khởi chạy chính của ứng dụng Angular.
-  **index.html** → File HTML chính.

 **angular.json** → Cấu hình của Angular CLI.

 **package.json** → Danh sách các package cần thiết.

---

## 4. Tạo Component Đầu Tiên

---

Angular chia ứng dụng thành nhiều **Component** (thành phần) nhỏ. Hãy tạo một component mới bằng lệnh:

```
bash
```

```
ng generate component hello
```

Hoặc viết ngắn gọn:

```
bash
```

```
ng g c hello
```

Lệnh này sẽ tạo ra thư mục  **hello/** bên trong **src/app/**, chứa các file:

- **hello.component.ts** → File TypeScript chứa logic.
- **hello.component.html** → File giao diện.
- **hello.component.css** → File CSS.
- **hello.component.spec.ts** → File kiểm thử.

## 5. Hiển thị Component

---

Mở **app.component.html** và thay đổi nội dung như sau:

```
html
```

```
<h1>Welcome to Angular!</h1>  
<app-hello></app-hello>
```

Sau đó, chỉnh sửa `hello.component.html`:

```
html
```

```
<p>Xin chào, đây là component đầu tiên của bạn!</p>
```

Lưu lại và chạy lại `ng serve`, bạn sẽ thấy component mới hiển thị.

---

## Bài tập thực hành

---

1. Cài đặt Angular CLI và tạo một dự án mới.
2. Chạy dự án với `ng serve` và kiểm tra giao diện mặc định.
3. Tạo một component mới có tên **"greeting"** hiển thị dòng chữ **"Hello Angular Learner!"**.
4. Nhúng component **greeting** vào `app.component.html`.

👉 Hãy thực hành và báo cho tôi biết nếu có lỗi hoặc cần giải thích thêm nhé! 🚀

Đã xong

Tuyệt vời! 🎉 Bây giờ chúng ta sẽ tiếp tục với **Bài 2: Data Binding trong Angular**.

---

## Bài 2: Data Binding trong Angular

---

**Data Binding** là cách Angular kết nối dữ liệu giữa **component** (logic) và **template** (giao diện). Angular hỗ trợ **4 loại Data Binding** chính:

Loại Binding	Ký hiệu	Mô tả
Interpolation	<code>{{ value }}</code>	Hiển thị dữ liệu trong template
Property Binding	<code>[property] = "value"</code>	Gán giá trị vào thuộc tính của HTML
Event Binding	<code>(event) = "handler()"</code>	Bắt sự kiện từ người dùng
Two-way Binding	<code>[(ngModel)] = "value"</code>	Đồng bộ dữ liệu giữa model và UI

## 1 Interpolation ({{}}) - Hiển thị dữ liệu

Interpolation dùng để hiển thị giá trị từ TypeScript ra HTML.

- ◆ Thêm biến trong `hello.component.ts`:

```
typescript

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

@Component({
  selector: 'app-hello',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent {
  name: string = 'Angular';
}
```

- ◆ Sử dụng Interpolation trong `hello.component.html`:

```
html

<h2>Welcome to {{ name }}!</h2>
```



🔧 Kết quả: **Welcome to Angular!**

---

## 2 Property Binding ([ ]) - Gán giá trị vào thuộc tính HTML

---

Cách sử dụng:

```
html
```

```
<input [value]="name">
```

🔥 Giá trị của **name** sẽ tự động gán vào **input** mà không cần **value="Angular"**.

📌 So sánh với Interpolation:

- Interpolation (**{{ }}**) chỉ hiển thị giá trị.
- Property Binding (**[ ]**) gán trực tiếp giá trị vào thuộc tính HTML.

Ví dụ với **disabled** button:

```
html
```

```
<button [disabled]="isDisabled">Click me</button>
```

Trong **hello.component.ts**:

typescript

```
isDisabled = true;
```

👉 **Kết quả:** Button sẽ bị vô hiệu hóa.

---

### 3 Event Binding (()) - Bắt sự kiện từ người dùng

---

Event Binding dùng để bắt sự kiện như **click**, **input**, **keyup** và gọi hàm xử lý.

#### ♦ Ví dụ: Bắt sự kiện Click

html

```
<button (click)="sayHello()">Click me</button>
<p>{{ message }}</p>
```

#### ♦ Trong **hello.component.ts**

typescript

```
message: string = '';

sayHello() {
  this.message = 'Xin chào từ Angular!';
}
```

👉 **Kết quả:** Khi nhấn nút, đoạn văn bản hiển thị "Xin chào từ Angular!".

---

## 4 Two-way Binding ([ (ngModel) ]) - Đồng bộ dữ liệu

---

Two-way Binding giúp đồng bộ dữ liệu giữa UI và Component.

🔗 **Cần import `FormsModule` trước khi sử dụng:** Mở `app.module.ts` và thêm:

```
typescript
```

```
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [FormsModule]
})
```

### ♦ Sử dụng [(ngModel)] trong `hello.component.html`

```
html
```

```
<input [(ngModel)]="name">
<p>Chào, {{ name }}!</p>
```

👉 **Kết quả:** Khi bạn nhập dữ liệu vào input, giá trị của `name` thay đổi theo.

---

## 🚀 Bài tập thực hành

---

1. Thêm một biến `age` vào `hello.component.ts` và hiển thị bằng Interpolation.
2. Thêm một button "**Hiển thị tuổi**", khi bấm sẽ hiển thị "`Tuổi của tôi là 25`" (sử dụng Event Binding).
3. Thêm một input để nhập tên người dùng, hiển thị tên đó bên dưới (sử dụng Two-way Binding).

💡 Hoàn thành xong, hãy báo cho tôi biết nhé! 🚀

không thấy App Module

Trong Angular, **App Module** nằm trong tệp `src/app/app.module.ts`. Đây là tệp chính để khai báo các module, component, và thư viện mà ứng dụng sử dụng.

## 1. Cấu trúc của `app.module.ts`

---

Khi bạn tạo một dự án Angular, `app.module.ts` sẽ tự động được tạo. Nếu bạn chưa thấy nó, hãy kiểm tra thư mục `src/app/`.

Mặc định, nội dung của `app.module.ts` sẽ như sau:

typescript

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

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

---

## 2. Thêm `FormsModule` vào App Module

---

Khi sử dụng **Two-way Binding** (`[(ngModel)]`), bạn cần **import** `FormsModule`.

**Cách thêm `FormsModule`:**

---

Mở `app.module.ts` và sửa lại như sau:

typescript

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms'; // Import FormsModule

import { AppComponent } from './app.component';
import { HelloComponent } from './hello/hello.component'; // Import
Component

@NgModule({
  declarations: [
    AppComponent,
    HelloComponent // Đăng ký HelloComponent vào Module
  ],
  imports: [
    BrowserModule,
    FormsModule // Thêm FormsModule vào imports
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

---

### 3. Vì sao cần **FormsModule**?

---

Khi bạn sử dụng [(ngModel)] trong Angular mà chưa import **FormsModule**, sẽ gặp lỗi:

python

Can't bind to 'ngModel' since it isn't a known property of 'input'

Bằng cách thêm **FormsModule**, Angular mới có thể hiểu và xử lý **Two-way Binding**.

---

 **Bài tập thực hành**

- 
1. Mở `app.module.ts` và thêm `FormsModule` vào `imports`.
  2. Chạy lại ứng dụng với `ng serve`.
  3. Kiểm tra lại `[(ngModel)]` trong component và đảm bảo không có lỗi.

👉 Xong rồi hãy báo cho tôi nhé! 🚀

không thấy `app.module.ts`

Nếu bạn không thấy `app.module.ts`, có thể bạn đã tạo dự án Angular ở chế độ **Standalone Components** (Angular từ phiên bản 14 trở đi cho phép tạo ứng dụng không cần `NgModule`).

---

## 1. Kiểm tra xem bạn có dùng Standalone Components không?

---

- Mở `src/main.ts` và kiểm tra xem có dòng này không:

```
typescript
```

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent);
```

- Nếu có, thì dự án của bạn đang sử dụng **Standalone Components**.

---

## 2. Cách xử lý cho dự án Standalone

---

Trong dự án **Standalone Components**, bạn không cần `app.module.ts`. Thay vào đó, bạn **import** `FormsModule` trực tiếp vào component.

Cách sửa trong `app.component.ts` hoặc `hello.component.ts`:

---

typescript

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

@Component({
  selector: 'app-hello',
  standalone: true, // Xác nhận đây là Standalone Component
  imports: [FormsModule], // Import FormsModule vào đây
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent {
  name = 'Angular Learner';
}
```

👉 Cách này sẽ giúp bạn dùng `[(ngModel)]` mà không cần `app.module.ts`.

---

### 3. Nếu bạn muốn tạo lại dự án có `app.module.ts`

---

Nếu bạn muốn tạo dự án theo kiểu **Module-Based**, khi chạy lệnh `ng new`, hãy đảm bảo chọn:

- ❌ **No** khi hỏi **"Would you like to enable Standalone Components?"**
- ✅ **Yes** cho Angular Routing và chọn CSS hoặc SCSS.

Nếu bạn đã lỡ chọn Standalone, có thể tạo lại dự án bằng lệnh:

bash

```
ng new my-angular-app --no-standalone
```

---

📌 **Bước tiếp theo**

---

✓ Nếu bạn đã sửa theo cách Standalone hoặc muốn tiếp tục với cách không cần `app.module.ts`, hãy thử lại bài tập **Two-way Binding** và báo cho tôi biết kết quả nhé! 🚀

Would you like to enable Standalone Components? -> chọn Yes thì có lợi ích gì

## ✓ Lợi ích của việc chọn "Yes" để sử dụng Standalone Components trong Angular

---

Từ phiên bản **Angular 14**, Google giới thiệu **Standalone Components**, cho phép xây dựng ứng dụng mà **không cần NgModule**. Khi bạn chọn **"Yes"** ở bước cài đặt, ứng dụng sẽ được cấu hình theo kiểu **Standalone**, mang lại một số lợi ích:

---

### 1. Loại bỏ sự phụ thuộc vào NgModule

---

- ◆ Trước đây, mỗi Component, Directive, Pipe đều phải được **khai báo trong một NgModule** (ví dụ: `app.module.ts`). Điều này đôi khi gây **rườm rà** và khó quản lý với các dự án lớn.
- ◆ Với **Standalone Components**, bạn có thể sử dụng trực tiếp **imports** ngay trong Component, không cần thông qua **NgModule**. Điều này giúp **giảm độ phức tạp** và làm cho Angular trở nên **giống React hơn**.

🔪 So sánh:

### ✗ Truyền thống (có NgModule)

---

typescript

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



## ✓ Standalone Components (Không cần NgModule)

---

typescript

```
// app.component.ts
import { Component } from '@angular/core';
import { bootstrapApplication } from '@angular/platform-browser';

@Component({
  selector: 'app-root',
  standalone: true, // Đây là một Standalone Component
  template: `<h1>Hello Angular!</h1>`,
})
export class AppComponent {}

// main.ts - Khởi động ứng dụng trực tiếp
bootstrapApplication(AppComponent);
```

🔥 **Kết quả:** Không cần `app.module.ts`, mà ứng dụng vẫn chạy bình thường.

---

## 2. Cải thiện hiệu suất và tốc độ khởi động

---

- Khi sử dụng **NgModule**, Angular cần **phân tích toàn bộ các Module**, làm cho quá trình khởi động chậm hơn.
  - Standalone Components giúp **giảm số lượng Module cần khởi tạo**, giúp ứng dụng khởi động nhanh hơn.
  - Dễ dàng tối ưu hóa **lazy loading**, vì mỗi component có thể được tải độc lập.
- 

## 3. Giảm Boilerplate Code

---

- Không cần tạo file `app.module.ts`.
- Không cần thêm component vào `declarations`.
- Không cần đăng ký module khi sử dụng component.

📌 **Ví dụ trước đây (cách truyền thống):**

typescript

```
@NgModule({
  declarations: [MyComponent],
  imports: [CommonModule],
  exports: [MyComponent]
})
export class MyModule { }
```

### Standalone Components (cách mới):

typescript

```
@Component({
  selector: 'app-my',
  standalone: true,
  template: `<p>Standalone Component!</p>`
})
export class MyComponent { }
```

 Nhìn gọn hơn rất nhiều! 

---

## 4. Dễ dàng quản lý Dependencies

---

Trước đây, khi sử dụng **HttpClientModule**, **FormsModule**, **RouterModule**,... bạn phải import chúng vào **app.module.ts**.

Với **Standalone Components**, bạn có thể import trực tiếp vào **Component**.

 Ví dụ: Import **FormsModule** trực tiếp vào **Component**

typescript

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

@Component({
  selector: 'app-hello',
  standalone: true,
  imports: [FormsModule], // Không cần `NgModule`
  template: `
    <input [(ngModel)]="name">
    <p>Chào, {{ name }}!</p>
  `,
})
export class HelloComponent {
  name = 'Angular Learner';
}
```

🔥 **Kết quả:** Không cần thêm **FormsModule** vào **NgModule**, mà vẫn dùng được `[(ngModel)]`.

---

## 5. Tích hợp tốt hơn với Micro-Frontend

---

- Standalone Components **rất phù hợp** khi xây dựng **Micro-Frontend** (ứng dụng chia thành nhiều phần nhỏ, có thể hoạt động độc lập).
  - Giảm độ phụ thuộc giữa các phần của ứng dụng.
- 

### 📌 Khi nào nên dùng Standalone Components?

---


- ✓ Nếu bạn xây dựng một **ứng dụng mới** từ Angular 14+ → **Nên dùng** Standalone Components.
  - ✓ Nếu bạn làm **Micro-Frontend** hoặc **Library riêng** → Standalone giúp dễ quản lý hơn.
  - ✓ Nếu bạn thích cách tiếp cận giống **React/Vue** hơn → Standalone là lựa chọn tốt.
- 

### 📌 Khi nào **KHÔNG** nên dùng Standalone Components?

---

- ✗ Nếu bạn đang làm việc với một dự án Angular cũ (**NgModule**-based), việc chuyển đổi sẽ **mất thời gian**.
- ✗ Nếu dự án của bạn có nhiều module phức tạp (Lazy Loading, Shared Modules), bạn

**cần cân nhắc** trước khi chuyển sang Standalone.

 **Tóm lại: Standalone Components là tương lai của Angular**, nhưng nếu bạn làm dự án lớn cũ, có thể chưa cần chuyển ngay.

---

## ♦ Tổng kết: Standalone Components có lợi ích gì?

---

### ✓ Lợi ích

Không cần **NgModule** → Code gọn hơn

Hiệu suất tốt hơn, khởi động nhanh hơn

Dễ dàng quản lý dependencies

Hỗ trợ tốt hơn cho Micro-Frontend

### ✗ Nhược điểm

Một số thư viện cũ chưa hỗ trợ

Không tương thích với dự án Angular cũ

Cần làm quen nếu đã quen với **NgModule**



-

 **Bạn đã chọn "Yes" cho Standalone Components → Bạn đang đi đúng hướng với Angular hiện đại!** 

---

## Bước tiếp theo

---

 Tiếp tục với **Component Communication (Truyền dữ liệu giữa các Component)** nhé! Bạn muốn tiếp tục ngay hay có câu hỏi nào cần giải đáp? 

Can't bind to 'ngModel' since it isn't a known property of 'input' và tôi đang dùng standalone component

## Giải quyết lỗi:

---

**Can't bind to 'ngModel' since it isn't a known property of 'input'**  
Lỗi này xảy ra vì bạn chưa **import **FormsModule**** vào **Standalone Component** của mình.

---

## ✓ Cách khắc phục

---

Trong **Standalone Components**, bạn không import **FormsModule** trong **app.module.ts** mà **phải import trực tiếp vào Component**.

### ♦ Cách sửa

---

Mở file **hello.component.ts** (hoặc component bạn đang dùng **[(ngModel)]**) và sửa lại như sau:

typescript

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms'; // ✓ Import FormsModule vào đây

@Component({
  selector: 'app-hello',
  standalone: true, // ✓ Đây là Standalone Component
  imports: [FormsModule], // ✓ Import FormsModule vào component
  template: `
    <input [(ngModel)]="name" placeholder="Nhập tên của bạn">
    <p>Chào, {{ name }}!</p>
  `,
})
export class HelloComponent {
  name = '';
}
```

---

## Giải thích

- **Standalone Components không có NgModule**, nên phải **import FormsModule** ngay trong **imports** của **Component**.
- Nếu không import, Angular sẽ không hiểu `[(ngModel)]` và gây ra lỗi.

---

## Bước tiếp theo

1. Sửa lại component như trên.
2. Chạy lại ứng dụng bằng **ng serve**.
3. Thử nhập giá trị vào input, kiểm tra xem dữ liệu có cập nhật không.

👉 Nếu vẫn có lỗi, hãy gửi code component của bạn để tôi giúp debug nhé! 🚀

làm sao để angular tự load lại sao mỗi lần save code

---

## Cách để Angular tự động reload sau mỗi lần lưu code

Khi bạn lập trình với Angular, **Angular CLI** có tính năng **tự động reload** trang web mỗi khi bạn chỉnh sửa code và lưu lại. Nếu bạn đang gặp vấn đề ứng dụng **không tự động reload**, hãy thử các cách sau:

## ✓ 1. Đảm bảo bạn đang chạy với **ng serve**

---

Angular tự động reload khi bạn dùng lệnh:

```
bash
```

```
ng serve --open
```

Hoặc đơn giản chỉ:

```
bash
```

```
ng serve
```

Sau đó, mở trình duyệt **http://localhost:4200/** và thử chỉnh sửa file **.ts**, **.html**, hoặc **.css** để kiểm tra.

## ✓ 2. Kiểm tra nếu **live reload** bị tắt

---

Nếu ứng dụng không reload tự động, hãy thử chạy:

```
bash
```

```
ng serve --live-reload
```

Hoặc:

```
bash
```

```
ng serve --live-reload=true
```

Điều này đảm bảo tính năng **Live Reload** được bật.

---

### ✓ 3. Kiểm tra **watch mode**

---

Đôi khi, Angular không theo dõi thay đổi trong file hệ thống. Hãy thử chạy:

```
bash
```

```
ng serve --poll=2000
```

Điều này giúp Angular kiểm tra file thay đổi sau mỗi **2 giây**.

---

### ✓ 4. Kiểm tra nếu bạn đang dùng **Standalone Components**

---

Nếu bạn đã dùng **Standalone Components**, chắc chắn rằng bạn đang khởi động ứng dụng bằng:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent);
```

Nếu `main.ts` của bạn chưa có cấu trúc như trên, có thể **Angular** không nhận diện đúng **Standalone Mode**.

---

## ✓ 5. Kiểm tra `angular.json`

---

Mở file `angular.json` và đảm bảo rằng phần cấu hình **"liveReload"** được bật:

json

```
"serve": {
  "options": {
    "liveReload": true,
    "poll": 2000
  }
}
```

Nếu `liveReload` bị tắt (`false`), hãy chỉnh sửa lại.

---

## ✓ 6. Nếu vẫn không được, thử xóa cache và `node_modules`

---

Thỉnh thoảng, lỗi do cache hoặc thư viện cũ. Thử xóa `node_modules` và cài lại dependencies:



```
bash
```

```
rm -rf node_modules package-lock.json  
npm install  
ng serve
```

---

## ✓ 7. Nếu dùng WSL hoặc Docker, thêm `--host`

---

Nếu bạn đang chạy Angular trên **WSL (Windows Subsystem for Linux)** hoặc trong **Docker**, hãy dùng:

```
bash
```

```
ng serve --host 0.0.0.0 --poll=2000
```

Điều này giúp Angular theo dõi thay đổi file trên môi trường Linux.

---

## Tóm lại

---

- ♦ Thử `ng serve --live-reload`
- ♦ Nếu không được, chạy `ng serve --poll=2000`
- ♦ Nếu vẫn lỗi, kiểm tra file `angular.json`
- ♦ Nếu đang dùng WSL hoặc Docker, thêm `--host 0.0.0.0`
- ♦ Nếu lỗi tiếp, xóa cache bằng `rm -rf node_modules package-lock.json` và cài lại

👉 Thử xong và báo lại tôi biết nếu vẫn còn lỗi nhé! 🚀

tôi đã thực hành xong

Tuyệt vời! 🎉 Bây giờ chúng ta sẽ tiếp tục với **Bài 3: Component Communication (Truyền dữ liệu giữa các Component)**.

---

## Bài 3: Component Communication trong Angular

Trong một ứng dụng Angular, các **Component** cần giao tiếp với nhau để chia sẻ dữ liệu. Có **4 cách chính** để truyền dữ liệu giữa các Component:

Cách truyền dữ liệu	Mô tả	Dùng khi nào?
<b>@Input()</b>	Truyền dữ liệu từ <b>Component cha</b> → <b>Component con</b>	Khi cha muốn gửi dữ liệu xuống con
<b>@Output() + EventEmitter</b>	Truyền dữ liệu từ <b>Component con</b> → <b>Component cha</b>	Khi con cần gửi sự kiện lên cha
<b>Service + Subject</b>	Dùng một <b>Service chung</b> để chia sẻ dữ liệu giữa nhiều Component	Khi nhiều Component không có quan hệ cha-con
<b>Local Storage / State Management</b>	Lưu dữ liệu vào <b>localStorage, Redux (NgRx)</b> để chia sẻ toàn bộ ứng dụng	Khi dữ liệu cần dùng ở nhiều nơi

### Truyền dữ liệu từ Component Cha → Component Con với **@Input()**

#### Ví dụ: Tạo Component con nhận dữ liệu từ cha

##### Bước 1: Tạo Component con (Child)

```
bash
```

```
ng generate component child
```

Hoặc viết tắt:

```
bash
```

```
ng g c child
```

Lệnh này sẽ tạo thư mục 📁 `child/` chứa các file cần thiết.

📌 **Bước 2:** Trong `child.component.ts`, thêm `@Input()`

```
typescript
```

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

@Component({
  selector: 'app-child',
  standalone: true,
  template: `<h3>Xin chào, {{ name }}!</h3>`,
})
export class ChildComponent {
  @Input() name: string = ''; // Nhận dữ liệu từ component cha
}
```

📌 **Bước 3:** Trong `app.component.ts`, tạo biến `parentName`

typescript

```
import { Component } from '@angular/core';
import { ChildComponent } from '../child/child.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [ChildComponent], // Import component con
  template: `
    <h1>Cha gọi con</h1>
    <app-child [name]="parentName"></app-child>
  `,
})
export class AppComponent {
  parentName = 'Angular Developer';
}
```

◆ **Kết quả:** Component con (**app-child**) sẽ hiển thị:

👉 "Xin chào, Angular Developer!"

---

## 2 Truyền dữ liệu từ Component Con → Component Cha với **@Output()**

---

### 🔧 Ví dụ: Component con gửi dữ liệu lên cha

---

👉 **Bước 1:** Trong **child.component.ts**, sử dụng **@Output()**

typescript

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

@Component({
  selector: 'app-child',
  standalone: true,
  template: `
    <button (click)="sendMessage()">Gửi tin nhắn</button>
  `,
})
export class ChildComponent {
  @Output() messageEvent = new EventEmitter<string>();

  sendMessage() {
    this.messageEvent.emit('Xin chào từ Component con!');
  }
}
```

 **Bước 2:** Trong `app.component.ts`, bắt sự kiện `messageEvent`

typescript

```
import { Component } from '@angular/core';
import { ChildComponent } from '../child/child.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [ChildComponent],
  template: `
    <h1>Cha nhận dữ liệu từ con</h1>
    <app-child (messageEvent)="receiveMessage($event)"></app-child>
    <p>{{ message }}</p>
  `,
})
export class AppComponent {
  message = '';

  receiveMessage(msg: string) {
    this.message = msg;
  }
}
```

- ◆ **Kết quả:** Khi bấm "Gửi tin nhắn", `messageEvent` sẽ gửi lên cha và hiển thị:  
👉 "Xin chào từ Component con!"

---

### 3 Truyền dữ liệu giữa nhiều Component với Service (`BehaviorSubject`)

---

Nếu hai component **không có quan hệ cha-con**, ta dùng **Service + Subject** để chia sẻ dữ liệu.

#### 🔧 Ví dụ: Dùng Service để chia sẻ dữ liệu giữa hai Component

---

##### 🔴 Bước 1: Tạo một Service

bash

ng g service data

Trong `data.service.ts`:

typescript

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

@Injectable({
  providedIn: 'root'
})
export class DataService {
  private messageSource = new BehaviorSubject<string>('Dữ liệu ban đầu');
  currentMessage = this.messageSource.asObservable();

  updateMessage(message: string) {
    this.messageSource.next(message);
  }
}
```

 **Bước 2:** Trong `Component 1`, gửi dữ liệu

typescript

```
import { Component } from '@angular/core';
import { DataService } from '../data.service';

@Component({
  selector: 'app-sender',
  standalone: true,
  template: `
    <button (click)="sendMessage()">Gửi dữ liệu</button>
  `,
})
export class SenderComponent {
  constructor(private dataService: DataService) {}

  sendMessage() {
    this.dataService.updateMessage('Dữ liệu mới từ SenderComponent!');
  }
}
```

 **Bước 3:** Trong **Component 2**, nhận dữ liệu



typescript

```
import { Component } from '@angular/core';
import { DataService } from '../data.service';

@Component({
  selector: 'app-receiver',
  standalone: true,
  template: `
    <p>Dữ liệu nhận được: {{ message }}</p>
  `,
})
export class ReceiverComponent {
  message: string = '';

  constructor(private dataService: DataService) {
    this.dataService.currentMessage.subscribe(msg => this.message = msg);
  }
}
```

◆ **Kết quả:** Khi nhấn nút trong **Component 1**, dữ liệu sẽ tự động cập nhật trong **Component 2**.

---

## Bài tập thực hành

---

1. **Tạo Component Cha và Component Con** (dùng **@Input()** để truyền tên từ cha xuống con).
2. **Tạo Component Con có nút "Gửi tin nhắn"**, dùng **@Output()** để gửi một tin nhắn lên cha.
3. **Tạo một Service** (**data.service.ts**) để chia sẻ dữ liệu giữa **hai Component không liên quan**.

---

👉 **Hãy thực hành và báo cho tôi biết kết quả nhé!** 🚀

tôi đã thực hành xong

Tuyệt vời! 🎉 Bạn đã nắm vững cách giao tiếp giữa các Component trong Angular. Bây giờ chúng ta sẽ tiếp tục với **Bài 4: Angular Routing (Điều hướng giữa các trang)**.

---

## Bài 4: Angular Routing (Điều hướng giữa các Component)

---

♦ **Routing** trong Angular giúp bạn tạo nhiều trang và chuyển đổi giữa chúng mà **không cần tải lại trang** (Single Page Application - SPA).

---

## 1 Cấu hình Routing trong Angular

---

- ♦ Khi bạn tạo dự án, nếu đã chọn **Enable Routing** (Would you like to add Angular routing? Yes), Angular đã tạo sẵn `app-routing.module.ts`.
  - ♦ Nếu bạn đang dùng **Standalone Components**, thì không có `app-routing.module.ts`. Bạn cần cấu hình routing trực tiếp trong `main.ts`.
- 

## 2 Tạo Component để điều hướng

---

Trước khi cấu hình Routing, bạn cần có các **Component đại diện cho các trang**.

📌 **Tạo 3 Component (Home, About, Contact):**

```
bash
```

```
ng g c home --standalone
ng g c about --standalone
ng g c contact --standalone
```

---

## 3 Cấu hình Router trong Standalone Components

---

Nếu bạn dùng **Standalone Components**, bạn cần cấu hình Routing trong `main.ts`.

Mở `main.ts` và chỉnh sửa như sau:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideRouter, Routes } from '@angular/router';
import { AppComponent } from './app/app.component';
import { HomeComponent } from './app/home/home.component';
import { AboutComponent } from './app/about/about.component';
import { ContactComponent } from './app/contact/contact.component';

const routes: Routes = [
  { path: '', component: HomeComponent }, // Trang chủ
  { path: 'about', component: AboutComponent }, // Trang Giới thiệu
  { path: 'contact', component: ContactComponent }, // Trang Liên hệ
];

bootstrapApplication(AppComponent, {
  providers: [provideRouter(routes)]
});
```

#### ♦ Giải thích:

- Khi vào `/`, Angular hiển thị **HomeComponent**.
- Khi vào `/about`, hiển thị **AboutComponent**.
- Khi vào `/contact`, hiển thị **ContactComponent**.

---

## Sử dụng `<router-outlet>` để hiển thị Component

---

 Mở `app.component.html` và chỉnh sửa:

html

```
<h1>Angular Routing Demo</h1>
<nav>
  <a routerLink="/">Trang Chủ</a> |
  <a routerLink="/about">Giới Thiệu</a> |
  <a routerLink="/contact">Liên Hệ</a>
</nav>

<!-- Đây là nơi hiển thị Component -->
<router-outlet></router-outlet>
```

---

## 5 Điều hướng bằng Code (Router Service)

---

Ngoài `<a routerLink="/">`, bạn có thể điều hướng bằng code trong **TypeScript**.

📌 Ví dụ: Điều hướng khi bấm nút

typescript

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

@Component({
  selector: 'app-home',
  standalone: true,
  template: `
    <h2>Trang chủ</h2>
    <button (click)="goToAbout()">Chuyển đến Giới thiệu</button>
  `,
})
export class HomeComponent {
  constructor(private router: Router) {}

  goToAbout() {
    this.router.navigate(['/about']);
  }
}
```

- ◆ Khi bấm nút, Angular sẽ điều hướng đến **/about**.

---

## Bài tập thực hành

---

1. Tạo thêm một trang "Services", cập nhật **routes** và thêm nút điều hướng.
2. Thử dùng Router Service (**router.navigate()**) để điều hướng giữa các trang.
3. Tìm hiểu cách dùng **routerLinkActive** để đánh dấu trang đang hoạt động.

👉 Hãy thực hành và báo tôi biết nếu có lỗi hoặc câu hỏi nhé! 🚀

không có `app-routing.module.ts` mà chỉ có các file này

Dựa vào ảnh bạn gửi, bạn **không có `app-routing.module.ts`**, điều này có thể là do bạn đang dùng **Standalone Components** hoặc dự án của bạn đã được thiết lập theo một cấu trúc khác.

---

## ✓ 1. Xác nhận cách dự án của bạn xử lý Routing

---

Bạn có một tệp **app.routes.ts**, điều này gợi ý rằng Routing có thể đã được định nghĩa trong đó. Hãy mở **app.routes.ts** và kiểm tra nội dung. Nếu nó chứa một danh sách **Routes**, bạn chỉ cần sử dụng nó.

- ♦ Nếu **app.routes.ts** có nội dung như sau:

typescript

```
import { Routes } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { AboutComponent } from '../about/about.component';

export const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent }
];
```

Thì đây chính là nơi **định nghĩa danh sách các tuyến đường (routes)** trong dự án của bạn.

---

## ✓ 2. Kiểm tra **main.ts** để xem Routing được cung cấp như thế nào

---

Mở **main.ts** và kiểm tra xem có **provideRouter(routes)** không. Nếu có, thì Routing của bạn đã được cấu hình theo cách mới (Standalone).

- ♦ Nếu **main.ts** của bạn trông như thế này:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideRouter } from '@angular/router';
import { AppComponent } from './app/app.component';
import { routes } from './app/app.routes';

bootstrapApplication(AppComponent, {
  providers: [provideRouter(routes)]
});
```

Thì bạn không cần **app-routing.module.ts**, vì Routing đã được cung cấp trực tiếp trong **main.ts**.

---

### ✓ 3. Thêm **<router-outlet>** vào **app.component.html**

---

Nếu bạn chưa có **<router-outlet>** trong **app.component.html**, hãy thêm nó để các Component được hiển thị đúng khi điều hướng:

html

```
<h1>Angular Routing Demo</h1>
<nav>
  <a routerLink="/">Trang Chủ</a> |
  <a routerLink="/about">Giới Thiệu</a>
</nav>

<router-outlet></router-outlet> <!-- Quan trọng -->
```

---

### Tóm lại

---

✓ **Bạn không cần `app-routing.module.ts`**, vì dự án của bạn đang dùng **Standalone Routing** với `app.routes.ts`.

✓ Routing của bạn có thể đã được định nghĩa trong `app.routes.ts`, và được **cung cấp trong `main.ts`**.

✓ Đảm bảo bạn có `<router-outlet>` trong `app.component.html` để hiển thị nội dung khi điều hướng.

👉 **Bạn hãy kiểm tra lại và báo cho tôi nếu cần hỗ trợ thêm nhé!** 🚀

hướng dẫn lại bài này dùng standalone component

## 📌 Bài 4: Angular Routing với Standalone Components

---

🔥 **Bạn đang sử dụng Standalone Components**, vì vậy **không cần `app-routing.module.ts`**. Thay vào đó, chúng ta sẽ sử dụng `app.routes.ts` để định nghĩa các tuyến đường và cấu hình Routing trong `main.ts`.

---

### 1 Tạo các Component cho từng trang

---

Chúng ta cần tạo các Component đại diện cho từng trang của ứng dụng.

📌 **Chạy lệnh để tạo Component Standalone:**

```
bash
```

```
ng g c home --standalone
ng g c about --standalone
ng g c contact --standalone
```

◆ **Lệnh trên tạo ra 3 Component (`home`, `about`, `contact`)** mà không cần `NgModule`.

---

### 2 Cấu hình Routes trong `app.routes.ts`

---

Mở file `app.routes.ts` và định nghĩa danh sách các tuyến đường:



typescript

```
import { Routes } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { AboutComponent } from '../about/about.component';
import { ContactComponent } from '../contact/contact.component';

export const routes: Routes = [
  { path: '', component: HomeComponent }, // Trang chủ
  { path: 'about', component: AboutComponent }, // Trang Giới thiệu
  { path: 'contact', component: ContactComponent }, // Trang Liên hệ
];
```

♦ **Giải thích:**

- Khi vào **/**, Angular sẽ hiển thị **HomeComponent**.
- Khi vào **/about**, sẽ hiển thị **AboutComponent**.
- Khi vào **/contact**, sẽ hiển thị **ContactComponent**.

---

### **Cấu hình Routing trong main.ts**

---

Mở **main.ts** và thêm **provideRouter(routes)**:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideRouter } from '@angular/router';
import { AppComponent } from '../app/app.component';
import { routes } from '../app/app.routes';

bootstrapApplication(AppComponent, {
  providers: [provideRouter(routes)]
});
```

♦ **Giải thích:**

`provideRouter(routes)`: Cung cấp danh sách các tuyến đường từ `app.routes.ts` để Angular biết cách điều hướng.

---

## 4 Cập nhật `app.component.html` để hiển thị Routing

---

Mở `app.component.html` và thêm `<router-outlet>` để hiển thị Component tương ứng với đường dẫn.

```
html
```

```
<h1>Angular Routing Demo</h1>

<nav>
  <a routerLink="/">Trang Chủ</a> |
  <a routerLink="/about">Giới Thiệu</a> |
  <a routerLink="/contact">Liên Hệ</a>
</nav>

<!-- Đây là nơi Component được hiển thị khi điều hướng -->
<router-outlet></router-outlet>
```

### ♦ Giải thích:

- `<router-outlet>`: Là nơi **Angular hiển thị Component tương ứng** với đường dẫn hiện tại.
- `<a routerLink="/">Trang Chủ</a>`: Điều hướng đến trang chủ mà không tải lại trang.

---

## 5 Điều hướng bằng Code (**Router Service**)

---

Ngoài cách dùng `<a routerLink="/">`, bạn có thể điều hướng bằng TypeScript.

### 📌 Ví dụ: Điều hướng khi bấm nút trong HomeComponent

Mở `home.component.ts` và sửa như sau:

typescript

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

@Component({
  selector: 'app-home',
  standalone: true,
  template: `
    <h2>Trang chủ</h2>
    <button (click)="goToAbout()">Chuyển đến Giới thiệu</button>
  `,
})
export class HomeComponent {
  constructor(private router: Router) {}

  goToAbout() {
    this.router.navigate(['/about']);
  }
}
```

#### ♦ Giải thích:

Khi bấm nút **"Chuyển đến Giới thiệu"**, Angular sẽ điều hướng đến **/about**.

---

## 6 Thêm **routerLinkActive** để làm nổi bật menu đang chọn

---

Mở **app.component.html** và sửa lại **nav** như sau:

html

```
<nav>
  <a routerLink="/" routerLinkActive="active">Trang Chủ</a> |
  <a routerLink="/about" routerLinkActive="active">Giới Thiệu</a> |
  <a routerLink="/contact" routerLinkActive="active">Liên Hệ</a>
</nav>
```

#### ♦ Giải thích:

- `routerLinkActive="active"` sẽ thêm class **"active"** vào link đang được chọn.
- Bạn có thể thêm CSS để làm nổi bật menu đang chọn:

CSS

```
.active {  
  font-weight: bold;  
  color: red;  
}
```

---

## Bài tập thực hành

---

1. Thêm một trang mới tên là **"Services"**, cập nhật `routes` và tạo menu điều hướng.
2. Thử dùng `router.navigate()` trong `about.component.ts` để điều hướng sang "Contact".
3. Dùng `routerLinkActive` để đánh dấu menu đang chọn.

---

👉 Hãy thực hành và báo lại tôi nếu có lỗi hoặc cần giải thích thêm nhé! 🚀

Property 'navigate' does not exist on type 'Router'. [plugin angular-compiler]

🔥 **Lỗi: Property 'navigate' does not exist on type 'Router'. [plugin angular-compiler]**

---

Lỗi này xảy ra có thể do:

1. Bạn chưa import `Router` từ `@angular/router`.
2. Bạn đang sử dụng `Router` mà chưa khai báo nó trong constructor.
3. Phiên bản Angular hoặc TypeScript bị lỗi (hiếm khi xảy ra).

---

## ✓ Cách khắc phục

---

#### ♦ Kiểm tra lại code trong Component của bạn

---

Hãy đảm bảo bạn đã import và khai báo `Router` đúng cách.

## 🔪 Đoạn code đúng (Cách sửa lỗi)

---

Mở **home.component.ts** (hoặc bất kỳ Component nào bạn đang dùng **navigate**) và sửa lại như sau:

typescript

```
import { Component } from '@angular/core';
import { Router } from '@angular/router'; // ✅ Import Router đúng cách

@Component({
  selector: 'app-home',
  standalone: true,
  template: `
    <h2>Trang chủ</h2>
    <button (click)="goToAbout()">Chuyển đến Giới thiệu</button>
  `,
})
export class HomeComponent {
  constructor(private router: Router) {} // ✅ Đúng cách khai báo Router

  goToAbout() {
    this.router.navigate(['/about']); // ✅ Sử dụng đúng cú pháp
  }
}
```

---

## ✅ Giải thích lỗi và cách sửa

---

### 1. Lỗi phổ biến: Quên import **Router**

- Nếu bạn không import **Router**, Angular sẽ không hiểu **navigate**.
- 💡 **Cách sửa:** Thêm dòng **import { Router } from '@angular/router';**.

### 2. Lỗi phổ biến: Quên khai báo **Router** trong constructor

- 💡 **Cách sửa:** Đảm bảo **constructor(private router: Router) {}** được khai báo đúng.

### 3. Lỗi do dùng **navigate()** sai cách

- **Sai:** **this.router.navigate('/about');**
- **Đúng:** **this.router.navigate(['/about']);**
  - 💡 **Giải thích:** **navigate()** nhận một mảng **['/about']**, không phải một chuỗi **'/about'**.

## Bước tiếp theo

---

1. Sửa lại Component theo hướng dẫn trên.
2. Chạy lại ứng dụng bằng **ng serve**.
3. Thử bấm nút điều hướng và kiểm tra kết quả.

👉 Nếu vẫn gặp lỗi, gửi tôi đoạn code của bạn để tôi giúp debug! 🚀

tôi đã thực hành xong

Tuyệt vời! 🎉 Bạn đã hoàn thành **Angular Routing với Standalone Components**. Bây giờ, chúng ta sẽ tiếp tục với **Bài 5: Reactive Forms trong Angular**.

---

## Bài 5: Reactive Forms trong Angular

---

**Reactive Forms** giúp bạn kiểm soát dữ liệu form, thực hiện validation, và quản lý trạng thái form một cách mạnh mẽ hơn so với Template-Driven Forms.

---

### 1 Cài đặt Reactive Forms

---

Vì bạn đang dùng **Standalone Components**, bạn cần **import** **ReactiveFormsModule** trực tiếp vào Component.

#### Cách import **ReactiveFormsModule**

---

Mở **app.component.ts** hoặc Component chứa form, thêm:

typescript

```
import { Component } from '@angular/core';
import { ReactiveFormsModule, FormGroup, FormControl, Validators } from
 '@angular/forms';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [ReactiveFormsModule], // ✓ Import ReactiveFormsModule trực tiếp
  templateUrl: './app.component.html',
})
export class AppComponent {
  loginForm = new FormGroup({
    username: new FormControl('', Validators.required),
    password: new FormControl('', [Validators.required,
Validators.minLength(6)])
  });

  onSubmit() {
    if (this.loginForm.valid) {
      console.log('Form Data:', this.loginForm.value);
    }
  }
}
```

---

## **Hiện thị Form trong HTML**

---

Mở **app.component.html** và thêm form:

html

```
<h2>Đăng nhập</h2>
<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
  <label for="username">Tên đăng nhập:</label>
  <input id="username" formControlName="username">
  <div *ngIf="loginForm.controls['username'].invalid &&
loginForm.controls['username'].touched">
    <small style="color: red">Tên đăng nhập không được để trống</small>
  </div>

  <label for="password">Mật khẩu:</label>
  <input id="password" type="password" formControlName="password">
  <div *ngIf="loginForm.controls['password'].invalid &&
loginForm.controls['password'].touched">
    <small style="color: red">Mật khẩu ít nhất 6 ký tự</small>
  </div>

  <button type="submit" [disabled]="loginForm.invalid">Đăng nhập</button>
</form>
```

---

### Giải thích

- ◆ `formGroup="loginForm"` → Liên kết Form với `FormGroup` trong TypeScript.
- ◆ `formControlName="username"` → Liên kết input với `FormControl`.
- ◆ `Validators.required` → Không cho phép để trống.
- ◆ `Validators.minLength(6)` → Mật khẩu tối thiểu 6 ký tự.
- ◆ `[disabled]="loginForm.invalid"` → Nút chỉ kích hoạt khi Form hợp lệ.

---

### Bài tập thực hành

1. Thêm một input "Email" vào Form, dùng `Validators.email`.
2. Hiện thị thông báo lỗi khi Email sai định dạng.
3. In dữ liệu Form ra console khi Submit.

👉 Hãy thực hành và báo cho tôi nếu có lỗi hoặc cần hỗ trợ nhé! 🚀

Can't bind to 'ngIf' since it isn't a known property of 'div' (used in the '\_AppComponent' component template).



🔥 **Lỗi: Can't bind to 'ngIf' since it isn't a known property of 'div'**

---

Lỗi này xảy ra vì **Angular không nhận diện \*ngIf**, nguyên nhân là do bạn chưa import **CommonModule** vào Standalone Component của bạn.

---

## ✓ Cách khắc phục

---

Trong **Standalone Components**, bạn **cần import CommonModule trực tiếp** vào **imports**.

## 📌 Sửa lại **app.component.ts**

---

Mở **app.component.ts** và sửa lại như sau:

typescript

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common'; // ✓ Import CommonModule
import { ReactiveFormsModule, FormGroup, FormControl, Validators } from
'@angular/forms';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule], // ✓ Import CommonModule để
dùng *ngIf
  templateUrl: './app.component.html',
})
export class AppComponent {
  loginForm = new FormGroup({
    username: new FormControl('', Validators.required),
    password: new FormControl('', [Validators.required,
Validators.minLength(6)])
  });

  onSubmit() {
    if (this.loginForm.valid) {
      console.log('Form Data:', this.loginForm.value);
    }
  }
}
```

## ✓ Giải thích

- **CommonModule** chứa các directive quan trọng của Angular như `*ngIf`, `*ngFor`, v.v.
- Khi dùng **Standalone Components**, bạn **phải import** **CommonModule** vào **imports** của **Component** để dùng `*ngIf`.
- **Nếu không import**, bạn sẽ gặp lỗi:

```
python
```

```
Can't bind to 'ngIf' since it isn't a known property of 'div'
```

---

## Bước tiếp theo

---

1. Sửa lại `app.component.ts` bằng cách thêm **CommonModule**.
2. Chạy lại ứng dụng bằng `ng serve`.
3. Thử kiểm tra xem `*ngIf` đã hoạt động đúng chưa.

👉 Nếu vẫn gặp lỗi, gửi tôi đoạn code để tôi giúp debug nhé! 🚀

tôi đã thực hành xong

Tuyệt vời! 🎉 Bạn đã thành thạo **Reactive Forms** trong Angular với **Standalone Components**.

Bây giờ, chúng ta sẽ tiếp tục với **Bài 6: Gọi API trong Angular bằng HttpClient**. 🚀

---

## Bài 6: Gọi API trong Angular với HttpClient

---

- ♦ **HttpClient** là một module trong Angular giúp bạn **gửi HTTP request** để **gọi API** từ **server**.
- ♦ Chúng ta sẽ thực hành **gọi API** giả từ `jsonplaceholder.typicode.com` để lấy danh sách người dùng.

---

### Import **HttpClientModule** vào Standalone Component

---

Vì bạn đang dùng **Standalone Components**, cần import **HttpClientModule** trực tiếp vào `imports`.

---

### Cách import **HttpClientModule**

---

Mở **app.component.ts** và thêm:

typescript

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClient, HttpClientModule } from '@angular/common/http';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, HttpClientModule], // ✔ Import HttpClientModule
  templateUrl: './app.component.html',
})
export class AppComponent {
  users: any[] = []; // Biến lưu danh sách người dùng

  constructor(private http: HttpClient) {} // ✔ Inject HttpClient vào constructor

  getUsers() {
    this.http.get('https://jsonplaceholder.typicode.com/users')
      .subscribe((data: any) => {
        this.users = data;
      });
  }
}
```

---

## **Hiện thị dữ liệu từ API trong HTML**

---

Mở **app.component.html** và sửa lại:

html

```
<h2>Danh sách người dùng</h2>
<button (click)="getUsers()">Tải danh sách</button>

<ul>
  <li *ngFor="let user of users">
    {{ user.name }} - {{ user.email }}
  </li>
</ul>
```

---

### Giải thích

---

- ◆ Import **HttpClientModule** → Giúp Angular có thể gửi HTTP request.
- ◆ Inject **HttpClient** trong constructor → Dùng để gọi API.
- ◆ Dùng **this.http.get()** để gọi API → Kết quả được lưu vào biến **users**.
- ◆ Dùng **\*ngFor** để hiển thị danh sách người dùng.
- ◆ Nút "Tải danh sách" → Khi bấm, Angular gọi API và hiển thị dữ liệu.

---

### Bài tập thực hành

---

1. **Thêm một API khác:** Gọi API **/posts** để hiển thị danh sách bài viết.
2. **Thêm input để nhập ID người dùng**, sau đó **gọi API** để lấy thông tin chi tiết của người đó.
3. **Hiển thị trạng thái tải dữ liệu (loading... khi đang gọi API).**

👉 **Hãy thực hành và báo tôi nếu có lỗi hoặc cần giải thích thêm nhé!** 🚀

TypeError: egetOrCreateAngularServerApp is not a function

---

🔥 **Lỗi: `TypeError: egetOrCreateAngularServerApp is not a function`**

---

Lỗi này thường xảy ra khi **phiên bản của Angular và các package khác không tương thích** hoặc bạn đang sử dụng **HttpClientModule** trong môi trường **Standalone Component** không đúng cách.

---

### 1. Kiểm tra và Cập nhật Dependencies

---

## **Bước 1: Kiểm tra phiên bản Angular**

---

Chạy lệnh sau để kiểm tra phiên bản Angular trong dự án của bạn:

```
bash
```

```
ng version
```

Nếu bạn thấy Angular CLI và các package **có sự chênh lệch lớn về phiên bản**, hãy đồng bộ lại.

## **Bước 2: Cập nhật tất cả các package**

---

Nếu dự án của bạn đang dùng phiên bản cũ hoặc không đồng bộ, hãy cập nhật bằng lệnh:

```
bash
```

```
ng update
```

Hoặc để cập nhật tất cả các package:

```
bash
```

```
ng update @angular/core @angular/cli
```

Nếu bạn sử dụng **Angular 15+**, hãy đảm bảo **package.json** của bạn có:

json

```
"dependencies": {  
  "@angular/common": "^15.0.0",  
  "@angular/core": "^15.0.0",  
  "@angular/forms": "^15.0.0",  
  "@angular/platform-browser": "^15.0.0",  
  "@angular/router": "^15.0.0"  
}
```

Sau đó chạy:

bash

```
rm -rf node_modules package-lock.json  
npm install
```

---

## ✓ 2. Đảm bảo Import **HttpClientModule** Đúng Cách

---

Trong **Standalone Components**, bạn cần import **HttpClientModule** đúng cách.

- ♦ Sửa lại **app.component.ts**:

typescript

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClient, HttpClientModule } from '@angular/common/http';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, HttpClientModule], // ✓ Import HttpClientModule
  đúng cách
  templateUrl: './app.component.html',
})
export class AppComponent {
  users: any[] = [];

  constructor(private http: HttpClient) {} // ✓ Inject HttpClient

  getUsers() {
    this.http.get('https://jsonplaceholder.typicode.com/users')
      .subscribe((data: any) => {
        this.users = data;
      });
  }
}
```

---

### ✓ 3. Kiểm tra **main.ts**

---

Mở file **main.ts** và đảm bảo bạn có **provideHttpClient()**.

- ♦ **Cách sửa:**

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideHttpClient } from '@angular/common/http';
import { provideRouter } from '@angular/router';
import { AppComponent } from './app/app.component';
import { routes } from './app/app.routes';

bootstrapApplication(AppComponent, {
  providers: [provideRouter(routes), provideHttpClient()] // ✓ Cung cấp
  HttpClient
});
```

#### ♦ Giải thích:

- `provideHttpClient()` là cách mới để thêm `HttpClient` trong `Standalone Components`.
- Nếu bạn không khai báo, bạn sẽ gặp lỗi khi gọi API.

---

### Bước tiếp theo

1. Sửa lại `main.ts` để dùng `provideHttpClient()`.
2. Sửa `app.component.ts` để import `HttpClientModule` đúng cách.
3. Chạy lại dự án bằng `ng serve` và kiểm tra.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn xảy ra nhé! 🚀

'HttpClientModule' is deprecated.ts(6385) index.d.ts(3758, 4): The declaration was marked as deprecated here.

### Lỗi: 'HttpClientModule' is deprecated.ts(6385)

Lỗi này xuất hiện vì **Angular 15+** đã thay đổi cách import `HttpClient`. Thay vì dùng `HttpClientModule`, bây giờ bạn cần sử dụng `provideHttpClient()`.

---

### ✓ Cách sửa lỗi

#### ♦ Cách cũ (Deprecated - Không nên dùng)

---



typescript

```
import { HttpClientModule } from '@angular/common/http'; // ❌ Không nên dùng
```

### ♦ Cách mới (Angular 15+ - Nên dùng)

---

#### Bước 1: Sửa **main.ts** để cung cấp HttpClient đúng cách

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideHttpClient } from '@angular/common/http'; // ✅ Dùng
provideHttpClient()
import { provideRouter } from '@angular/router';
import { AppComponent } from './app/app.component';
import { routes } from './app/app.routes';

bootstrapApplication(AppComponent, {
  providers: [
    provideRouter(routes),
    provideHttpClient() // ✅ Cung cấp HttpClient theo cách mới
  ]
});
```

**Bước 2: Xóa **HttpClientModule** khỏi **app.component.ts**** Trong Component của bạn (ví dụ **app.component.ts**), chỉ cần import **HttpClient**, KHÔNG import **HttpClientModule**.

#### ♦ Sửa lại **app.component.ts**:

typescript

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http'; // ✔ Không cần
HttpClientModule nữa

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  users: any[] = [];

  constructor(private http: HttpClient) {} // ✔ HttpClient vẫn hoạt động

  getUsers() {
    this.http.get('https://jsonplaceholder.typicode.com/users')
      .subscribe((data: any) => {
        this.users = data;
      });
  }
}
```

---

## ✔ Giải thích

- **HttpClientModule** đã bị **deprecated** vì Angular 15+ cung cấp một cách mới là **provideHttpClient()**.
- **provideHttpClient()** được thêm vào **providers** trong **bootstrapApplication()**, giúp Angular xử lý **HTTP requests** mà không cần import module vào từng Component.
- Component chỉ cần **inject HttpClient**, không cần **HttpClientModule**.

---

## 🚀 Bước tiếp theo

1. Xóa **HttpClientModule** khỏi **app.component.ts**.
2. Thêm **provideHttpClient()** vào **main.ts**.
3. Chạy lại ứng dụng bằng **ng serve** và thử gọi API.

👉 Hãy thử sửa và báo tôi nếu vẫn còn lỗi nhé! 🚀

```
ERROR Error [NullInjectorError]:  
R3InjectorError(Standalone[_AppComponent])[_HttpClient -> _HttpClient ->  
_HttpClient]:
```

🔥 **Lỗi: `R3InjectorError(Standalone[_AppComponent])[_HttpClient -> _HttpClient -> _HttpClient]`**

---

Lỗi này xảy ra vì **Angular chưa được cung cấp `HttpClient`**, do bạn chưa thêm `provideHttpClient()` trong `main.ts`.

---

## ✓ Cách sửa lỗi

---

Bạn cần đảm bảo `provideHttpClient()` đã được thêm vào `bootstrapApplication()` trong `main.ts`.

### ♦ Bước 1: Sửa lại `main.ts`

---

Mở `main.ts` và sửa lại như sau:

```
typescript
```

```
import { bootstrapApplication } from '@angular/platform-browser';  
import { provideHttpClient } from '@angular/common/http'; // ✓ Thêm  
HttpClient  
import { provideRouter } from '@angular/router';  
import { AppComponent } from './app/app.component';  
import { routes } from './app/app.routes';  
  
bootstrapApplication(AppComponent, {  
  providers: [  
    provideRouter(routes),  
    provideHttpClient() // ✓ Cung cấp HttpClient  
  ]  
});
```

### ♦ Bước 2: Đảm bảo `HttpClientModule` không còn trong Component

---

Mở `app.component.ts` (hoặc Component đang dùng `HttpClient`) và kiểm tra xem bạn có đang `import HttpClientModule` không.

**Nếu có, hãy xóa nó đi, chỉ giữ lại `HttpClient`.**

## ♦ Sửa lại `app.component.ts`

typescript

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http'; // ✓ Chỉ import
HttpClient, KHÔNG import HttpClientModule

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  users: any[] = [];

  constructor(private http: HttpClient) {} // ✓ Inject HttpClient đúng cách

  getUsers() {
    this.http.get('https://jsonplaceholder.typicode.com/users')
      .subscribe((data: any) => {
        this.users = data;
      });
  }
}
```

---

## ✓ Giải thích lỗi

- Lỗi xảy ra vì `HttpClient` chưa được cung cấp trong `bootstrapApplication()`.
- Cách khắc phục: Thêm `provideHttpClient()` vào `providers` trong `main.ts`.
- Không cần `HttpClientModule` nữa, vì cách mới dùng `provideHttpClient()`.

---

## Bước tiếp theo

1. Sửa lại `main.ts`, đảm bảo có `provideHttpClient()`.
2. Xóa `HttpClientModule` khỏi tất cả Component.
3. Chạy lại ứng dụng bằng `ng serve`.
4. Thử gọi API và kiểm tra xem có hoạt động không.

👉 Hãy thử sửa và báo tôi nếu vẫn còn lỗi nhé! 🚀

```
bootstrapApplication(AppComponent, { providers: [provideRouter(routes),
provideHttpClient()] }).catch((err) => console.error(err)); vẫn lỗi
```

Nếu bạn đã thêm `provideHttpClient()` vào `bootstrapApplication()` trong `main.ts`, nhưng vẫn gặp lỗi `R3InjectorError(Standalone[_AppComponent])[_HttpClient -> _HttpClient -> _HttpClient]`, hãy thử các bước khắc phục sau:

---

## ✓ 1. Kiểm tra lại import của `HttpClient` trong Component

---

Mở `app.component.ts` hoặc bất kỳ Component nào đang dùng `HttpClient`, đảm bảo **chỉ import `HttpClient`, không import `HttpClientModule`**.

- ♦ Sửa lại `app.component.ts`:

typescript

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http'; // ✓ Chỉ import
HttpClient

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  users: any[] = [];

  constructor(private http: HttpClient) {} // ✓ Inject HttpClient đúng cách

  getUsers() {
    this.http.get('https://jsonplaceholder.typicode.com/users')
      .subscribe((data: any) => {
        this.users = data;
      });
  }
}
```

 **KHÔNG import `HttpClientModule` trong Component**, nếu có thì hãy xóa nó đi.

---

## ✓ 2. Kiểm tra lại `main.ts`

---

Mở `main.ts` và đảm bảo bạn có `provideHttpClient()`, giống như sau:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideHttpClient } from '@angular/common/http';
import { provideRouter } from '@angular/router';
import { AppComponent } from './app/app.component';
import { routes } from './app/app.routes';

bootstrapApplication(AppComponent, {
  providers: [
    provideRouter(routes),
    provideHttpClient() // ✅ Cung cấp HttpClient
  ]
}).catch((err) => console.error(err));
```

 Chắc chắn rằng bạn **KHÔNG** import `HttpClientModule` ở bất kỳ đâu.

---

### ✅ 3. Kiểm tra xem bạn có nhiều `bootstrapApplication()` không

---

Nếu bạn vô tình có nhiều `bootstrapApplication()` trong các file khác nhau, hãy xóa bớt đi.

Chạy lệnh sau để tìm kiếm:

bash

```
grep -r "bootstrapApplication" src/
```

Nếu thấy nhiều hơn một file chứa `bootstrapApplication()`, hãy đảm bảo chỉ có một nơi duy nhất là `main.ts`.

---

## ✓ 4. Xóa cache và cài lại dependencies

---

Nếu đã sửa tất cả nhưng vẫn lỗi, có thể Angular bị lỗi cache. Chạy các lệnh sau:

```
bash
```

```
rm -rf node_modules package-lock.json  
npm install  
ng serve
```

---

### Tóm lại:

- ✓ Chỉ import **HttpClient**, không import **HttpClientModule** trong Component.
- ✓ Đảm bảo **provideHttpClient()** có trong **bootstrapApplication()** trong **main.ts**.
- ✓ Kiểm tra xem có nhiều **bootstrapApplication()** không.
- ✓ Xóa cache và cài lại dependencies nếu cần.

👉 Hãy thử các bước này và báo tôi nếu vẫn còn lỗi nhé! 🚀

làm sao để debug trong angular

### Cách Debug trong Angular

---

Khi làm việc với Angular, bạn có thể sử dụng nhiều phương pháp khác nhau để debug lỗi, bao gồm: **Console Log**, **Debugger**, **Breakpoints**, **Angular DevTools**, **Network Request Debugging**, và **Error Handling**.

---

#### ✓ 1. Debug bằng **console.log()** (Cách cơ bản nhất)

Phương pháp dễ nhất để kiểm tra lỗi là sử dụng **console.log()** để in ra giá trị của biến hoặc trạng thái của ứng dụng.

📌 Ví dụ: Kiểm tra dữ liệu API trong Component

typescript

```
getUsers() {
  this.http.get('https://jsonplaceholder.typicode.com/users')
    .subscribe((data: any) => {
      console.log('Dữ liệu API:', data); // ✔ Kiểm tra dữ liệu trả về
      this.users = data;
    }, (error) => {
      console.error('Lỗi khi gọi API:', error); // ✔ Kiểm tra lỗi nếu có
    });
}
```

- ♦ Mở **Developer Tools (F12)** → **Console** để xem log.

---

## ✔ 2. Debug bằng **debugger**; (Dừng code tại một điểm)

---

Dùng từ khóa **debugger**; để tạm dừng chương trình khi chạy trong Chrome DevTools.

📌 **Ví dụ:** Dừng code khi gọi API

typescript

```
getUsers() {
  debugger; // 🚫 Khi chạy đến đây, chương trình sẽ dừng lại
  this.http.get('https://jsonplaceholder.typicode.com/users')
    .subscribe((data: any) => {
      this.users = data;
    });
}
```

- ♦ Khi chạy code, mở **DevTools (F12)** → **Sources** → **Resume (F8)** để tiếp tục thực thi.

---

## ✔ 3. Debug bằng Breakpoints trong Chrome DevTools

---



Bạn có thể đặt **breakpoints** trực tiếp trong trình duyệt Chrome để dừng mã nguồn TypeScript.

#### 📌 Cách sử dụng Breakpoints:

1. Mở Chrome DevTools (F12).
2. Chọn tab **Sources**.
3. Điều hướng đến file TypeScript trong thư mục **webpack://** hoặc **ng://** (dành cho Angular).
4. Đặt Breakpoint bằng cách **click vào số dòng**.
5. **Làm mới trang (F5)** → Angular sẽ dừng code tại điểm đó.
6. Sử dụng **Step Over (F10)** hoặc **Step Into (F11)** để debug từng dòng.

---

## ✓ 4. Debug Service và Dependency Injection

Nếu Service không hoạt động đúng, bạn có thể kiểm tra **dịch vụ có được Inject đúng không**.

#### 📌 Ví dụ: Kiểm tra Service trong Component

typescript

```
constructor(private dataService: DataService) {  
  console.log('DataService Injected:', this.dataService);  
}
```

♦ Nếu thấy **undefined**, có thể bạn chưa **đăng ký Service trong provide hoặc NgModule**.

---

## ✓ 5. Debug Network Request (Kiểm tra API Call)

Nếu ứng dụng gọi API nhưng **không có dữ liệu hoặc bị lỗi**, hãy kiểm tra bằng **tab Network trong DevTools**.

#### 📌 Cách kiểm tra:

1. Mở **Chrome DevTools (F12)** → **Network** → **Fetch/XHR**.
2. Tải lại trang (F5) hoặc bấm nút gọi API.

### 3. Kiểm tra:

- **Status Code** (200 OK, 404 Not Found, 500 Internal Server Error).
- **Request URL**: Đúng API chưa?
- **Response Body**: API trả về gì?
- **Headers**: Có lỗi CORS hoặc xác thực không?

♦ Nếu API trả về lỗi **401 (Unauthorized)**, kiểm tra **JWT Token** hoặc **Authentication Headers**.

---

## ✓ 6. Dùng Angular DevTools (Debug UI, Change Detection)

---

Angular có một extension cho Chrome gọi là **Angular DevTools**, giúp debug Component, Change Detection, Router...

### 📌 Cách cài đặt:

1. Cài đặt **Angular DevTools** tại [Chrome Web Store](#).
2. Mở DevTools (**F12**) → Chọn **Angular** tab.
3. Kiểm tra Component, Inputs, Outputs, Change Detection...

♦ Nếu UI không cập nhật khi thay đổi dữ liệu, kiểm tra Change Detection trong Angular DevTools.

---

## ✓ 7. Debug Lỗi **NullInjectorError** (Service Injection)

---

Nếu gặp lỗi **NullInjectorError**, bạn có thể debug bằng cách kiểm tra Service có được **cung cấp (providedIn)** đúng không.

### 📌 Kiểm tra trong Service:

typescript

```
@Injectable({
  providedIn: 'root' // ✓ Kiểm tra nếu service được cung cấp toàn cục
})
export class MyService { }
```

📌 Hoặc kiểm tra **main.ts** nếu dùng **provideHttpClient()**

typescript

```
bootstrapApplication(AppComponent, {  
  providers: [provideHttpClient()] // ✓ Nếu đang debug HttpClient  
});
```

---

## ✓ 8. Debug lỗi **ExpressionChangedAfterItHasBeenCheckedError**

---

Lỗi này xảy ra khi dữ liệu bị thay đổi sau khi Angular đã render UI.

🔧 Cách debug lỗi này:

- Sử dụng `setTimeout(() => {...})` để trì hoãn thay đổi.
- Dùng `ChangeDetectorRef.detectChanges()` để cập nhật thủ công.

Ví dụ:

typescript

```
constructor(private cdr: ChangeDetectorRef) {}  
  
ngAfterViewInit() {  
  setTimeout(() => {  
    this.someValue = 'Updated!';  
    this.cdr.detectChanges(); // ✓ Giúp tránh lỗi  
    ExpressionChangedAfterItHasBeenCheckedError  
  });  
}
```

---

## ✓ 9. Debug Routing (Router Navigation)

---

Nếu điều hướng (Navigation) không hoạt động đúng, hãy kiểm tra:

- **routerLink** có đúng không?
- Router có được cấu hình đúng trong **main.ts** không?

### Kiểm tra Navigation trong Component

typescript

```
import { Router } from '@angular/router';

constructor(private router: Router) {}

goToAbout() {
  console.log('Điều hướng đến /about');
  this.router.navigate(['/about']);
}
```

### Nếu lỗi Navigation, kiểm tra trong DevTools → Console

python

Cannot match any routes. URL Segment: 'about'

- ♦ Nếu gặp lỗi này, kiểm tra lại **routes** trong **app.routes.ts**.

---

## 10. Debug bằng **catchError** trong Observable

---

Khi gọi API hoặc xử lý dữ liệu, bạn có thể **bắt lỗi** bằng **catchError** trong RxJS.

### Ví dụ: Bắt lỗi khi gọi API

typescript

```
import { catchError } from 'rxjs/operators';
import { of } from 'rxjs';

getUsers() {
  this.http.get('https://jsonplaceholder.typicode.com/users')
    .pipe(
      catchError(error => {
        console.error('Lỗi khi gọi API:', error);
        return of([]); // ✓ Trả về mảng rỗng nếu API lỗi
      })
    )
    .subscribe(data => {
      this.users = data;
    });
}
```

---

## Tóm lại: Cách Debug Angular hiệu quả

---

### Phương pháp

**Console Log**

**Debugger**

**Breakpoints**

**Network Debugging**

**Angular DevTools**

**NullInjectorError**

**ExpressionChangedAfterItHasBeenCheckedError**

**Routing Debugging**

### Cách thực hiện

Dùng `console.log()` để kiểm tra dữ liệu

Dùng `debugger;` để dừng code

Đặt breakpoints trong Chrome DevTools

Kiểm tra API Call trong tab **Network**

Cài đặt Chrome Extension để debug Component

Kiểm tra `providedIn: 'root'` trong Service

Dùng `setTimeout()` hoặc `detectChanges()`


Kiểm tra `router.navigate()` và `routerLink`

---

## RxJS Error Handling

Dùng `catchError` để bắt lỗi API

### Bước tiếp theo

1. Hãy thử debug vấn đề bạn đang gặp phải.
2. Nếu vẫn có lỗi, gửi tôi đoạn code và lỗi cụ thể, tôi sẽ giúp bạn debug nhanh chóng! 

đã xong

Tuyệt vời!  Bạn đã nắm vững các kỹ thuật debug trong Angular và có thể xử lý các lỗi một cách hiệu quả.

### Bạn muốn tiếp tục học gì tiếp theo?

- 1 State Management với NgRx (Quản lý trạng thái ứng dụng)
- 2 Lazy Loading Modules (Tối ưu hiệu suất với Lazy Load)
- 3 Unit Testing trong Angular (Viết Test với Jasmine & Karma)
- 4 Angular Performance Optimization (Tối ưu hiệu suất)
- 5 Angular SSR với Angular Universal (SEO và Server-Side Rendering)

 Bạn muốn đi tiếp với chủ đề nào? 

tiếp tục với bài tiếp theo

## Bài 7: Quản lý trạng thái trong Angular với NgRx (State Management)

- ♦ NgRx là thư viện giúp quản lý trạng thái ứng dụng trong Angular, tương tự Redux trong React.
- ♦ Dùng NgRx giúp ứng dụng hoạt động mượt mà hơn, dễ debug hơn, đặc biệt với ứng dụng lớn.
- ♦ Mô hình NgRx hoạt động theo Flux Pattern:

Action → Reducer → Store → Selector → Component.

### 1. Cài đặt NgRx

 Chạy lệnh sau để cài đặt NgRx vào dự án Angular:

bash

```
ng add @ngrx/store  
ng add @ngrx/effects  
ng add @ngrx/store-devtools
```

#### ♦ Giải thích:

- **@ngrx/store**: Quản lý trạng thái.
- **@ngrx/effects**: Xử lý các side effects (gọi API, async actions).
- **@ngrx/store-devtools**: Dùng để debug bằng Redux DevTools.

---

## ✓ 2. Cấu trúc thư mục của NgRx

---

- ♦ Trong thư mục **src/app**, tạo thư mục **store/** và các file:

CSS

```
src/app/store/  
├─ state.ts      (Định nghĩa trạng thái)  
├─ actions.ts   (Định nghĩa Actions)  
├─ reducer.ts   (Định nghĩa Reducers)  
└─ selectors.ts (Định nghĩa Selectors)
```

---

## ✓ 3. Tạo State (**state.ts**)

---

- 📌 Mở file **store/state.ts** và định nghĩa trạng thái ban đầu:

typescript

```
export interface AppState {  
  count: number;  
}  
  
export const initialState: AppState = {  
  count: 0  
};
```

♦ **Giải thích:**

- **AppState** chứa một biến **count** để đếm số.
- **initialState** là trạng thái mặc định khi ứng dụng khởi chạy.

---

## ✓ 4. Tạo Actions (**actions.ts**)

---

✂ **Mở file **store/actions.ts** và tạo các Action:**

typescript

```
import { createAction } from '@ngrx/store';  
  
export const increment = createAction('[Counter] Increment');  
export const decrement = createAction('[Counter] Decrement');  
export const reset = createAction('[Counter] Reset');
```

♦ **Giải thích:**

- **increment**: Tăng **count** lên 1.
- **decrement**: Giảm **count** đi 1.
- **reset**: Đặt lại **count** về 0.



## ✓ 5. Tạo Reducer (**reducer.ts**)

---

📌 Mở file **store/reducer.ts** để cập nhật state dựa trên Action:

typescript

```
import { createReducer, on } from '@ngrx/store';
import { increment, decrement, reset } from '../actions';
import { AppState, initialState } from '../state';

export const counterReducer = createReducer(
  initialState,
  on(increment, (state) => ({ count: state.count + 1 })),
  on(decrement, (state) => ({ count: state.count - 1 })),
  on(reset, () => initialState)
);
```

### ♦ Giải thích:

- **createReducer()** tạo một reducer quản lý trạng thái.
- **on(increment, (state) => {...})** xử lý khi action **increment** được gọi.

## ✓ 6. Tạo Selectors (**selectors.ts**)

---

📌 Mở file **store/selectors.ts** để lấy dữ liệu từ Store:

typescript

```
import { createSelector } from '@ngrx/store';
import { AppState } from '../state';

export const selectCount = createSelector(
  (state: AppState) => state,
  (state) => state.count
);
```

♦ Giải thích:

`selectCount` lấy giá trị `count` từ Store.

---

## ✓ 7. Cấu hình Store trong `main.ts`

---

✚ Mở `main.ts` và thêm `provideStore()`:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [provideStore({ count: counterReducer })]
});
```

♦ Giải thích:

`provideStore({ count: counterReducer })`: Đăng ký `counterReducer` vào Store.

---

## ✓ 8. Sử dụng NgRx trong Component (`app.component.ts`)

---

✚ Mở `app.component.ts` và sửa lại để sử dụng NgRx:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { increment, decrement, reset } from '../store/actions';
import { selectCount } from '../store/selectors';

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$ = this.store.select(selectCount);

  constructor(private store: Store) {}

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

♦ **Giải thích:**

- `this.count$ = this.store.select(selectCount);` → Lấy giá trị `count` từ Store.
- `this.store.dispatch(increment());` → Gửi Action `increment` đến Store.

---

## ✓ 9. Hiện thị Counter trong Template (`app.component.html`)

---

✂ **Mở `app.component.html` và sửa lại để hiện thị số đếm:**

html

```
<h1>Counter NgRx</h1>
<p>Giá trị hiện tại: {{ count$ | async }}</p>

<button (click)="increment()">Tăng</button>
<button (click)="decrement()">Giảm</button>
<button (click)="reset()">Reset</button>
```

♦ **Giải thích:**

`{{ count$ | async }}` → Lấy giá trị `count` từ Store.

---

## ✓ 10. Debug với Redux DevTools

📌 **Cài đặt Redux DevTools để dễ dàng debug Store:**

bash

```
ng add @ngrx/store-devtools
```

Mở **Chrome DevTools > Redux** để xem **state** và **action history**.

---

## 🚀 Bài tập thực hành

1. Thêm một action `multiply` để nhân giá trị `count` với 2.
2. Thêm một input cho người dùng nhập số, sau đó nhấn "Cộng số này" để cộng vào `count`.
3. Tìm hiểu `@ngrx/effects` để gọi API khi nhấn nút.

👉 **Hãy thực hành và báo tôi nếu có lỗi hoặc câu hỏi nhé!** 🚀

Property 'store' is used before its initialization.

## 🔥 Lỗi: Property 'store' is used before its initialization.

Lỗi này xảy ra vì Angular yêu cầu khai báo biến trong constructor trước khi sử dụng.

### ✓ Cách sửa lỗi

🔧 Trong `app.component.ts`, đảm bảo `store` được inject đúng cách:

#### ◆ Cách đúng:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { increment, decrement, reset } from './store/actions';
import { selectCount } from './store/selectors';

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$ = this.store.select(selectCount); // ❌ Lỗi: store chưa được khởi tạo

  constructor(private store: Store) { // ✓ Inject `store` trước khi sử dụng
    this.count$ = this.store.select(selectCount); // ✓ Đúng: gọi trong constructor
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

♦ **Giải thích:**

- Lỗi xảy ra do `count$ = this.store.select(selectCount);` được gọi trước khi Angular khởi tạo `store`.
- Cách sửa: Di chuyển `count$ = this.store.select(selectCount);` vào `constructor`.

---

✓ **Cách khác: Khởi tạo bằng `undefined`**

---

Nếu bạn muốn giữ `count$` ở trên, có thể gán giá trị mặc định:

typescript

```
count$ = undefined as any; // ✓ Tránh lỗi
```

Hoặc dùng `| null`:

typescript

```
count$: Observable<number> | null = null;
```

Sau đó, trong `constructor()`, gán lại giá trị:

typescript

```
this.count$ = this.store.select(selectCount);
```

---

## Bước tiếp theo

---

1. Sửa lại **app.component.ts** theo cách trên.
2. Chạy lại ứng dụng bằng **ng serve**.
3. Kiểm tra Redux DevTools để xem state có thay đổi đúng không.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn còn! 🚀

No overload matches this call. Overload 1 of 9, '(mapFn: (state: object) => number): Observable<number>', gave the following error. Argument of type 'MemoizedSelector<AppState, number, (s1: AppState) => number>' is not assignable to parameter of type '(state: object) => number'. Types of parameters 'state' and 'state' are incompatible. Property 'count' is missing in type '{} but required in type 'AppState'. Overload 2 of 9, '(key: never): Observable<never>', gave the following error. Argument of type 'MemoizedSelector<AppState, number, (s1: AppState) => number>' is not assignable to parameter of type 'never'.ts(2769) state.ts(2, 5): 'count' is declared here. (alias) const selectCount: MemoizedSelector<AppState, number, (s1: AppState) => number>

## Lỗi: No overload matches this call

---

Lỗi này xảy ra vì **Store** không nhận diện đúng **AppState**, có thể do **Store** chưa được khởi tạo đúng cách trong **provideStore()**, hoặc **Reducer** không đúng.

---

## Cách sửa lỗi

---

### ◆ Bước 1: Cập nhật **state.ts**

---

Mở **state.ts** và đảm bảo rằng **AppState** được định nghĩa đúng:

typescript

```
export interface AppState {  
  count: number;  
}  
  
export const initialState: AppState = {  
  count: 0  
};
```

✓ **Đảm bảo rằng `initialState` chứa tất cả các thuộc tính của `AppState`.**

---

## ◆ **Bước 2: Sửa `selectors.ts`**

---

✚ **Mở `selectors.ts` và sửa lại `selectCount` như sau:**

typescript

```
import { createFeatureSelector, createSelector } from '@ngrx/store';  
import { AppState } from '../state';  
  
export const selectAppState = createFeatureSelector<AppState>('appState');  
  
export const selectCount = createSelector(  
  selectAppState,  
  (state) => state.count  
);
```

## ◆ **Giải thích lỗi trước đó:**

- **Lỗi xảy ra vì Angular không tìm thấy `count` trong Store.**
  - **Cách sửa:** Dùng `createFeatureSelector<AppState>('appState')` để đảm bảo Angular biết `AppState` nằm ở đâu trong Store.
-



### ◆ Bước 3: Cấu hình `provideStore()` trong `main.ts`

---

📌 Mở `main.ts` và đảm bảo Store được cấu hình đúng:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ appState: counterReducer }) // ✅ Đặt đúng key 'appState'
  ]
});
```

✅ Đảm bảo key `appState` trong `provideStore()` khớp với `createFeatureSelector<AppState>('appState')`.

---

### ◆ Bước 4: Sửa `app.component.ts`

---

📌 Mở `app.component.ts` và sửa lại cách lấy `count$`:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from '../store/actions';
import { selectCount } from '../store/selectors';
import { AppState } from '../store/state';

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>;

  constructor(private store: Store<AppState>) { // ✓ Khai báo Store với
    đúng type
    this.count$ = this.store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

- ✓ Đảm bảo rằng **Store** được khởi tạo với **AppState** (**Store<AppState>**).
- ✓ Dùng **Observable<number>** để đảm bảo **count\$** nhận giá trị đúng.

---

## ✓ Bước tiếp theo

---

1. Sửa lại các file như hướng dẫn trên.
2. Chạy lại ứng dụng bằng **ng serve**.
3. Mở **Redux DevTools** trong **Chrome** để kiểm tra **state**.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn còn! 🚀

```
Property 'store' is used before its initialization.ts(2729)  
app.component.ts(63, 15): 'store' is declared here. (property)  
AppComponent.store: Store<object> No quick fixes available
```

### 🔥 Lỗi: **Property 'store' is used before its initialization.ts(2729)**

---

Lỗi này xuất hiện vì bạn đang sử dụng **store** trước khi Angular kịp khởi tạo nó trong **constructor**.

---

### ✓ Cách sửa lỗi

---

#### ◆ Cách 1: Khởi tạo **store** trong Constructor

---

Hãy đảm bảo rằng **store** chỉ được sử dụng sau khi nó đã được khởi tạo trong **constructor()**.

📌 Sửa lại **app.component.ts** như sau:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from '../store/actions';
import { selectCount } from '../store/selectors';
import { AppState } from '../store/state';

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>; // ✔ Dùng `!` để tránh lỗi undefined

  constructor(private store: Store<AppState>) {
    this.count$ = this.store.select(selectCount); // ✔ Chỉ sử dụng store
    sau khi nó được khởi tạo
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

### ✔ Giải thích:

- **count\$: Observable<number>;** → Dùng **!** để TypeScript hiểu rằng biến này sẽ được khởi tạo sau.
- Di chuyển **this.count\$ = this.store.select(selectCount);** vào **constructor()** để đảm bảo **store** đã được inject trước khi sử dụng.

---

### ♦ Cách 2: Định nghĩa **store** trong Constructor

---

Một cách khác là **không khai báo store** như một **property** của **class**, mà chỉ sử dụng nó bên trong **constructor()**.

📌 Ví dụ:

typescript

```
export class AppComponent {
  count$: Observable<number>;

  constructor(private readonly store: Store<AppState>) {
    this.count$ = store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }
}
```

🚀 **Lợi ích:** Không còn lỗi **store is used before its initialization** vì **store** chỉ được sử dụng trong **constructor**.

---

### ♦ Cách 3: Kiểm tra **provideStore()** trong **main.ts**

---

Nếu lỗi vẫn xảy ra, có thể là do **Store** chưa được khởi tạo đúng cách trong **main.ts**.

🔧 Kiểm tra **main.ts** và sửa lại nếu cần:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ appState: counterReducer }) // ✔ Đảm bảo key 'appState'
    đúng
  ]
});
```

✔ Đảm bảo rằng key trong `provideStore()` khớp với key trong `createFeatureSelector()` trong `selectors.ts`.

---

## ✔ Bước tiếp theo

---

1. Sửa lại `app.component.ts` theo cách 1 hoặc cách 2.
2. Chạy lại ứng dụng bằng `ng serve`.
3. Kiểm tra Redux DevTools để xem state có thay đổi đúng không.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn còn! 🚀

```
X [ERROR] NG8004: No pipe found with name 'async'. [plugin angular-
compiler] src/app/app.component.html:49:33: 49 | <p>Giá trị hiện tại: {{
count$ | async }}</p> | ~~~~~ Error occurs in the template of component
AppComponent. src/app/app.component.ts:59:15: 59 | templateUrl:
'./app.component.html', | ~~~~~
```

🔥 **Lỗi: No pipe found with name 'async'**

---

Lỗi này xảy ra vì `AsyncPipe` (`| async`) cần `CommonModule`, nhưng nó chưa được import vào `app.component.ts` khi bạn sử dụng **Standalone Components**.

---

## ✔ Cách sửa lỗi

---

🔪 **Thêm `CommonModule` vào imports của `AppComponent`**

Mở `app.component.ts` và sửa lại như sau:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from './store/actions';
import { selectCount } from './store/selectors';
import { AppState } from './store/state';
import { CommonModule } from '@angular/common'; // ✓ Import CommonModule

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule], // ✓ Import CommonModule để dùng async pipe
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>;

  constructor(private store: Store<AppState>) {
    this.count$ = this.store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

---

## ✓ Giải thích

---

- **async** là một **pipe** có sẵn trong **CommonModule**.
  - Vì bạn đang dùng **Standalone Components**, bạn phải **import CommonModule trực tiếp** trong **imports** của Component.
  - Nếu **không import CommonModule**, Angular **không nhận diện được async pipe**, dẫn đến lỗi **No pipe found with name 'async'**.
-

## Bước tiếp theo

---

1. Sửa lại `app.component.ts` để import `CommonModule`.
2. Chạy lại ứng dụng bằng `ng serve`.
3. Kiểm tra xem giá trị `count$ | async` có hiển thị đúng không.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn còn! 🚀

@ngrx/store: The feature name "appState" does not exist in the state, therefore createFeatureSelector cannot access it. Be sure it is imported in a loaded module using `StoreModule.forRoot('appState', ...)` or `StoreModule.forFeature('appState', ...)`. If the default state is intended to be undefined, as is the case with router state, this development-only warning message can be ignored. ERROR TypeError: Cannot read properties of undefined (reading 'count') at i0 (C:\Tuan\Development\Angular-Practice-Base-On-Document\my-angular-app\src\app\store\selectors.ts:8:20)

### Lỗi: "The feature name 'appState' does not exist in the state"

---

Lỗi này xảy ra vì:

1. `provideStore({ appState: counterReducer })` trong `main.ts` không đúng.
2. Trong `createFeatureSelector<AppState>('appState')`, key `'appState'` không tồn tại trong Store.
3. Reducer hoặc `AppState` không được khai báo đúng.

## Cách sửa lỗi

---

### ◆ Bước 1: Kiểm tra `provideStore()` trong `main.ts`

---

Mở `main.ts` và sửa lại như sau:



typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ count: counterReducer }) // ✔ Đặt key là 'count' thay vì 'appState'
  ]
});
```

#### ♦ Lý do sửa lỗi:

- **Ban đầu:** `provideStore({ appState: counterReducer })` tạo Store với key `'appState'`.
- **Sửa lại:** Thay `'appState'` thành `'count'`, vì reducer của chúng ta chỉ quản lý `count`.

---

#### ♦ Bước 2: Kiểm tra `state.ts`

---

📌 Mở `state.ts` và đảm bảo `AppState` được khai báo đúng:

typescript

```
export interface AppState {
  count: number;
}

export const initialState: AppState = {
  count: 0
};
```

✓ Đảm bảo `initialState` chứa tất cả thuộc tính của `AppState`.

---

### ◆ Bước 3: Sửa `selectors.ts` để dùng `createFeatureSelector()` đúng cách

---

✚ Mở `selectors.ts` và sửa lại như sau:

typescript

```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { AppState } from '../state';

// ✓ Thay đổi 'AppState' thành 'count' để khớp với `provideStore()`
export const selectAppState = createFeatureSelector<AppState>('count');

export const selectCount = createSelector(
  selectAppState,
  (state) => state.count
);
```

#### ◆ Lý do sửa lỗi:

- `createFeatureSelector<AppState>('AppState')` không tìm thấy `'AppState'` trong Store.
- **Sửa lại:** Dùng `'count'`, vì `provideStore({ count: counterReducer })` đang sử dụng `'count'` làm key.

### ◆ Bước 4: Sửa `app.component.ts`

---

✚ Mở `app.component.ts` và sửa lại như sau:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from '../store/actions';
import { selectCount } from '../store/selectors';
import { AppState } from '../store/state';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule], // ✓ Import CommonModule để dùng async pipe
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>;

  constructor(private store: Store<AppState>) {
    this.count$ = this.store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

#### ✓ Giải thích:

- Giữ **Store<AppState>** để đảm bảo TypeScript hiểu kiểu dữ liệu của Store.
- Sử dụng **selectCount** đúng key đã khai báo trong **selectors.ts**.

---

#### ✓ Tóm lại: Các bước sửa lỗi

1. Sửa **main.ts** để dùng **provideStore({ count: counterReducer })**.

2. Sửa `selectors.ts` để dùng `createFeatureSelector<AppState>('count')`.
3. Kiểm tra `state.ts` để đảm bảo `AppState` chứa `count`.
4. Kiểm tra `app.component.ts` để chắc chắn rằng `store.select(selectCount)` đang trỏ đến `count`.
5. Chạy lại ứng dụng bằng `ng serve` và kiểm tra trên Redux DevTools.

👉 Hãy thử sửa và báo tôi nếu lỗi vẫn còn! 🚀

```
app.component.html:49 ERROR TypeError: Cannot read properties of
undefined (reading 'count') at selectors.ts:8:20
```

### 🔥 Lỗi: `Cannot read properties of undefined (reading 'count')`

Lỗi này xảy ra vì **Angular không tìm thấy `count` trong Store**, có thể do một trong các nguyên nhân sau:

1. Key trong `provideStore()` không khớp với `createFeatureSelector()`.
2. Reducer chưa khởi tạo đúng hoặc State chưa có giá trị ban đầu.
3. Component đang cố truy cập vào Store trước khi nó được khởi tạo.

## ✓ Cách sửa lỗi

### ◆ Bước 1: Đảm bảo `provideStore()` trong `main.ts` đúng

📌 Mở `main.ts` và kiểm tra lại:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ appState: counterReducer }) // ✓ Đặt key là 'appState'
  ]
});
```

### ◆ Lý do:

- Đảm bảo `appState` là key trong Store.
  - Nếu bạn đã đặt `provideStore({ count: counterReducer })` trước đó, hãy sửa lại thành `{ appState: counterReducer }`.
- 

## ♦ Bước 2: Sửa `state.ts` để đảm bảo State có giá trị ban đầu

---

📌 Mở `state.ts` và đảm bảo `AppState` có đầy đủ dữ liệu:

typescript

```
export interface AppState {  
  count: number;  
}  
  
export const initialState: AppState = {  
  count: 0 // ✅ Đảm bảo có giá trị ban đầu  
};
```

### ♦ Lý do:

Nếu `count` không có giá trị ban đầu, **Store sẽ bị undefined**, gây lỗi khi truy cập `state.count`.

---

## ♦ Bước 3: Kiểm tra `reducer.ts`

---

📌 Mở `reducer.ts` và kiểm tra cách xử lý State:

typescript

```
import { createReducer, on } from '@ngrx/store';
import { increment, decrement, reset } from './actions';
import { AppState, initialState } from './state';

export const counterReducer = createReducer(
  initialState,
  on(increment, (state) => ({ ...state, count: state.count + 1 })),
  on(decrement, (state) => ({ ...state, count: state.count - 1 })),
  on(reset, () => initialState)
);
```

#### ♦ Lý do:

- Đảm bảo mỗi `on()` luôn trả về một object mới với `count` được cập nhật.
- Tránh viết sai kiểu `{ count: state.count + 1 }` (không bao gồm `...state`).

---

#### ♦ Bước 4: Kiểm tra `selectors.ts`

---

📌 Mở `selectors.ts` và kiểm tra cách lấy State:

typescript

```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { AppState } from './state';

// ✔ Đảm bảo key trong `provideStore()` khớp với key ở đây
export const selectAppState = createFeatureSelector<AppState>('appState');

export const selectCount = createSelector(
  selectAppState,
  (state) => state.count
);
```

#### ♦ Lý do:

- Nếu `createFeatureSelector<AppState>('count')` không khớp với `provideStore({ appState: counterReducer })`, sẽ bị lỗi `undefined`.
  - Đảm bảo dùng `'appState'` nếu bạn đặt `provideStore({ appState: counterReducer })`.
- 

#### ♦ Bước 5: Kiểm tra `app.component.ts`

---

📌 Mở `app.component.ts` và đảm bảo `store.select(selectCount)` đang lấy giá trị đúng:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from '../store/actions';
import { selectCount } from '../store/selectors';
import { AppState } from '../store/state';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule], // ✓ Import CommonModule để dùng async pipe
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>;

  constructor(private store: Store<AppState>) {
    this.count$ = this.store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

#### ♦ Lý do:

- Đảm bảo `Store<AppState>` có kiểu dữ liệu chính xác.
- `count$ = this.store.select(selectCount);` chỉ truy cập `state.count` sau khi `store` đã được khởi tạo.

---

✓ **Bước tiếp theo**

---



1. Sửa lại `main.ts` để dùng `provideStore({ appState: counterReducer })`.
2. Sửa lại `selectors.ts` để dùng `createFeatureSelector<AppState>('appState')`.
3. Kiểm tra `state.ts` để đảm bảo `count` có giá trị mặc định.
4. Chạy lại ứng dụng bằng `ng serve`.
5. Kiểm tra Redux DevTools để xem state có khởi tạo đúng không.

👉 Hãy thử sửa và báo tôi nếu lỗi vẫn còn! 🚀

```
RROR TypeError: Cannot read properties of undefined (reading 'count') at
selectors.ts:8:20 at nxgr-store.mjs:928:24 at memoized (nxgr-
store.mjs:804:33) at defaultStateFn (nxgr-store.mjs:832:39) at nxgr-
store.mjs:931:30 at memoized (nxgr-store.mjs:804:33) at nxgr-
store.mjs:704:44 at map.js:7:31 at OperatorSubscriber2._this._next
(OperatorSubscriber.js:14:9) at Subscriber2.next (Subscriber.js:32:12)
export const selectCount = createSelector( selectAppState, (state) =>
state.count );
```

### 🔥 Lỗi: Cannot read properties of undefined (reading 'count')

Lỗi này có nghĩa là `state` bị `undefined`, do đó khi bạn cố đọc `state.count`, Angular sẽ báo lỗi. Nguyên nhân có thể là:

1. `provideStore()` trong `main.ts` không khớp với `createFeatureSelector()`.
2. State ban đầu (`initialState`) không được khởi tạo đúng.
3. Reducer không xử lý đúng giá trị mặc định.
4. Component truy cập `store` trước khi Store sẵn sàng.

### ✓ Cách sửa lỗi

#### ◆ Bước 1: Kiểm tra `provideStore()` trong `main.ts`

🔪 Mở `main.ts` và đảm bảo bạn dùng đúng key:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ appState: counterReducer }) // ✓ Key phải trùng với
createFeatureSelector()
  ]
});
```

### ✓ Lý do:

Đảm bảo key 'appState' trong `provideStore()` trùng với key trong `selectors.ts`.

---

### ♦ Bước 2: Kiểm tra `selectors.ts`

---

✚ Mở `selectors.ts` và đảm bảo key trùng với `provideStore()`:

typescript

```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { AppState } from '../state';

// ✓ Key trong createFeatureSelector phải trùng với key trong
provideStore()
export const selectAppState = createFeatureSelector<AppState>('appState');

export const selectCount = createSelector(
  selectAppState,
  (state) => state ? state.count : 0 // ✓ Tránh lỗi nếu state chưa khởi tạo
);
```

### ✓ Lý do:

Nếu `state` là `undefined`, ta trả về `0` thay vì truy cập `state.count` gây lỗi.

---

### ◆ Bước 3: Kiểm tra `state.ts`

---

📌 Mở `state.ts` và đảm bảo `state` có giá trị mặc định:

typescript

```
export interface AppState {  
  count: number;  
}  
  
export const initialState: AppState = {  
  count: 0 // ✓ Đảm bảo có giá trị ban đầu  
};
```

### ✓ Lý do:

Nếu `initialState` không có giá trị, `state.count` có thể bị `undefined`.

---

### ◆ Bước 4: Kiểm tra `reducer.ts`

---

📌 Mở `reducer.ts` và đảm bảo `state` được xử lý đúng:

typescript

```
import { createReducer, on } from '@ngrx/store';
import { increment, decrement, reset } from '../actions';
import { AppState, initialState } from '../state';

export const counterReducer = createReducer(
  initialState,
  on(increment, (state) => ({ ...state, count: state.count + 1 })),
  on(decrement, (state) => ({ ...state, count: state.count - 1 })),
  on(reset, () => initialState)
);
```

#### ✓ Lý do:

- Sử dụng `...state` để bảo toàn trạng thái ban đầu.
- Tránh ghi đè toàn bộ `state` bằng `{ count: ... }` mà không có `...state`.

---

#### ◆ Bước 5: Kiểm tra `app.component.ts`

📌 Mở `app.component.ts` và đảm bảo cách lấy dữ liệu từ Store đúng:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from './store/actions';
import { selectCount } from './store/selectors';
import { AppState } from './store/state';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule], // ✓ Import CommonModule để dùng async pipe
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>;

  constructor(private store: Store<{ appState: AppState }>) {
    this.count$ = this.store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

### ✓ Lý do:

Đảm bảo kiểu dữ liệu của Store là `{ appState: AppState }`, trùng với key trong `provideStore()`.

### ✓ Bước tiếp theo

1. Sửa `main.ts` để dùng `provideStore({ appState: counterReducer })`.

2. Sửa `selectors.ts` để dùng `createFeatureSelector<AppState>('appState')`.
3. Sửa `state.ts` để đảm bảo state có giá trị mặc định.
4. Sửa `app.component.ts` để dùng `Store<{ appState: AppState }>` đúng cách.
5. Chạy lại ứng dụng bằng `ng serve`.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn còn! 🚀

app.component.html:49 @ngrx/store: The feature name "appState" does not exist in the state, therefore createFeatureSelector cannot access it. Be sure it is imported in a loaded module using StoreModule.forRoot('appState', ...) or StoreModule.forFeature('appState', ...). If the default state is intended to be undefined, as is the case with router state, this development-only warning message can be ignored.

### 🔥 Lỗi: "The feature name 'appState' does not exist in the state"

Lỗi này xảy ra vì key của state trong `provideStore()` không khớp với key được sử dụng trong `createFeatureSelector()`.

## ✓ Cách sửa lỗi

### ◆ Bước 1: Kiểm tra `provideStore()` trong `main.ts`

🔗 Mở `main.ts` và đảm bảo `provideStore()` được khai báo đúng:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ appState: counterReducer }) // ✓ Đặt key là 'appState'
  ]
});
```

### ◆ Lý do:

Key 'appState' trong `provideStore()` phải trùng với key trong `selectors.ts`.

---

## ♦ Bước 2: Kiểm tra `selectors.ts`

---

📌 Mở `selectors.ts` và đảm bảo `createFeatureSelector()` sử dụng đúng key:

typescript

```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { AppState } from './state';

// ✔ Key trong createFeatureSelector phải trùng với key trong
provideStore()
export const selectAppState = createFeatureSelector<AppState>('appState');

export const selectCount = createSelector(
  selectAppState,
  (state) => state.count
);
```

✔ Đảm bảo rằng:

Key 'appState' trong `createFeatureSelector<AppState>('appState')` trùng với key trong `provideStore()`.

---

## ♦ Bước 3: Kiểm tra `state.ts`

---

📌 Mở `state.ts` và đảm bảo State có giá trị ban đầu:

typescript

```
export interface AppState {  
  count: number;  
}  
  
export const initialState: AppState = {  
  count: 0 // ✓ Đảm bảo có giá trị ban đầu  
};
```

### ✓ Lý do:

Nếu state không có giá trị mặc định, `state.count` sẽ bị `undefined`.

---

### ♦ Bước 4: Kiểm tra `reducer.ts`

---

🔗 Mở `reducer.ts` và kiểm tra:

typescript

```
import { createReducer, on } from '@ngrx/store';  
import { increment, decrement, reset } from './actions';  
import { AppState, initialState } from './state';  
  
export const counterReducer = createReducer(  
  initialState,  
  on(increment, (state) => ({ ...state, count: state.count + 1 })),  
  on(decrement, (state) => ({ ...state, count: state.count - 1 })),  
  on(reset, () => initialState)  
);
```

### ✓ Lý do:

- Giữ `...state` để bảo toàn trạng thái ban đầu.
- Tránh ghi đè toàn bộ `state` bằng `{ count: ... }` mà không có `...state`.



---

## ♦ Bước 5: Kiểm tra `app.component.ts`

---

📌 Mở `app.component.ts` và đảm bảo Store đang truy cập đúng key:

typescript

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { increment, decrement, reset } from '../store/actions';
import { selectCount } from '../store/selectors';
import { AppState } from '../store/state';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule], // ✅ Import CommonModule để dùng async pipe
  templateUrl: './app.component.html',
})
export class AppComponent {
  count$: Observable<number>;

  constructor(private store: Store<{ appState: AppState }>) {
    this.count$ = this.store.select(selectCount);
  }

  increment() {
    this.store.dispatch(increment());
  }

  decrement() {
    this.store.dispatch(decrement());
  }

  reset() {
    this.store.dispatch(reset());
  }
}
```

✅ Lý do:

Đảm bảo kiểu dữ liệu của Store là `{ appState: AppState }`, trùng với key trong `provideStore()`.

---

## ✓ Tóm lại: Các bước sửa lỗi

---

1. Sửa `main.ts` để dùng `provideStore({ appState: counterReducer })`.
2. Sửa `selectors.ts` để dùng `createFeatureSelector<AppState>('appState')`.
3. Sửa `state.ts` để đảm bảo state có giá trị mặc định.
4. Sửa `app.component.ts` để dùng `Store<{ appState: AppState }>` đúng cách.
5. Chạy lại ứng dụng bằng `ng serve`.

👉 Hãy thử lại và báo tôi nếu lỗi vẫn còn! 🚀

việc đặt provider trong `app.config.ts` với `main.ts` có khác nhau gì

## 📌 Sự khác biệt giữa việc đặt `providers` trong `app.config.ts` và `main.ts` trong Angular

---

Cả `app.config.ts` và `main.ts` đều có thể được sử dụng để **cấu hình providers** cho ứng dụng Angular, nhưng chúng có một số **sự khác biệt quan trọng**.

## ✓ 1. Đặt `providers` trong `main.ts`

---

### Cách triển khai

---

Trước đây, tất cả `providers` đều được đặt trực tiếp trong `main.ts` như sau:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { counterReducer } from '../app/store/reducer';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    provideStore({ appState: counterReducer }) // ✓ Định nghĩa providers ở
    đây
  ]
});
```

## 📌 Ưu điểm

---

- Đơn giản, dễ hiểu, vì tất cả `providers` được khai báo trực tiếp khi bootstrap ứng dụng.
- Không cần tạo file config riêng.

## Nhược điểm

---

- Khi ứng dụng **quá lớn**, `main.ts` có thể trở nên **khó quản lý** do chứa quá nhiều cấu hình.
  - Khó tách biệt giữa **cấu hình ứng dụng** và **bootstrap logic**.
- 

## 2. Đặt `providers` trong `app.config.ts`

---

### Cách triển khai


---

Angular 15+ giới thiệu cách **tách biệt cấu hình ứng dụng** bằng cách đặt `providers` trong `app.config.ts`:

 Tạo file `app.config.ts` và đặt `providers` vào đây:

typescript

```
import { ApplicationConfig } from '@angular/core';
import { provideStore } from '@ngrx/store';
import { counterReducer } from './store/reducer';

export const appConfig: ApplicationConfig = {
  providers: [
    provideStore({ appState: counterReducer }) //  Đặt providers trong appConfig
  ]
};
```

 Sau đó, import `appConfig` vào `main.ts`:

typescript

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';
import { appConfig } from './app/app.config';

bootstrapApplication(AppComponent, appConfig);
```

## Ưu điểm

- **Tách biệt logic cấu hình ra khỏi `main.ts`**, giúp dễ đọc và dễ quản lý hơn.
- **Dễ mở rộng**: Khi ứng dụng lớn hơn, có thể dễ dàng thêm hoặc sửa đổi cấu hình mà không ảnh hưởng đến phần bootstrap.
- **Chuẩn hóa theo Angular hiện đại (15+)**.

## Nhược điểm

Cần thêm một file `app.config.ts` (không phải vấn đề lớn, nhưng có thể gây khó khăn với người mới bắt đầu).

## Khi nào nên dùng cách nào?

♦ Trường hợp	✓ Dùng <code>main.ts</code>	✓ Dùng <code>app.config.ts</code>
Ứng dụng nhỏ	✓ Dùng <code>main.ts</code> để đơn giản	✗ Không cần thiết
Ứng dụng lớn	✗ Quá nhiều <code>providers</code> làm <code>main.ts</code> rối	✓ Dùng <code>app.config.ts</code> để dễ quản lý
Code theo Angular hiện đại	✗ Cách cũ, không được khuyến khích	✓ Chuẩn hóa theo Angular 15+
Muốn mở rộng dễ dàng	✗ Khó mở rộng nếu <code>main.ts</code> phức tạp	✓ Có thể dễ dàng tách config

## Kết luận

- Nếu ứng dụng nhỏ, bạn có thể đặt `providers` trực tiếp trong `main.ts`.

- Nếu ứng dụng lớn, đặt `providers` trong `app.config.ts` sẽ giúp dễ quản lý, dễ mở rộng và theo chuẩn Angular mới.
- Angular 15+ khuyến khích dùng `app.config.ts` thay vì `main.ts` để giữ cho `main.ts` sạch hơn.

👉 Bạn đang làm dự án lớn hay nhỏ? Bạn muốn dùng cách nào? 🚀

ChatGPT can make mistakes. Check important info.