



Angular 6

Làm sao để học tốt khóa học angular 6

Làm các theo các demo cơ bản

Làm tất cả các bài tập được giao

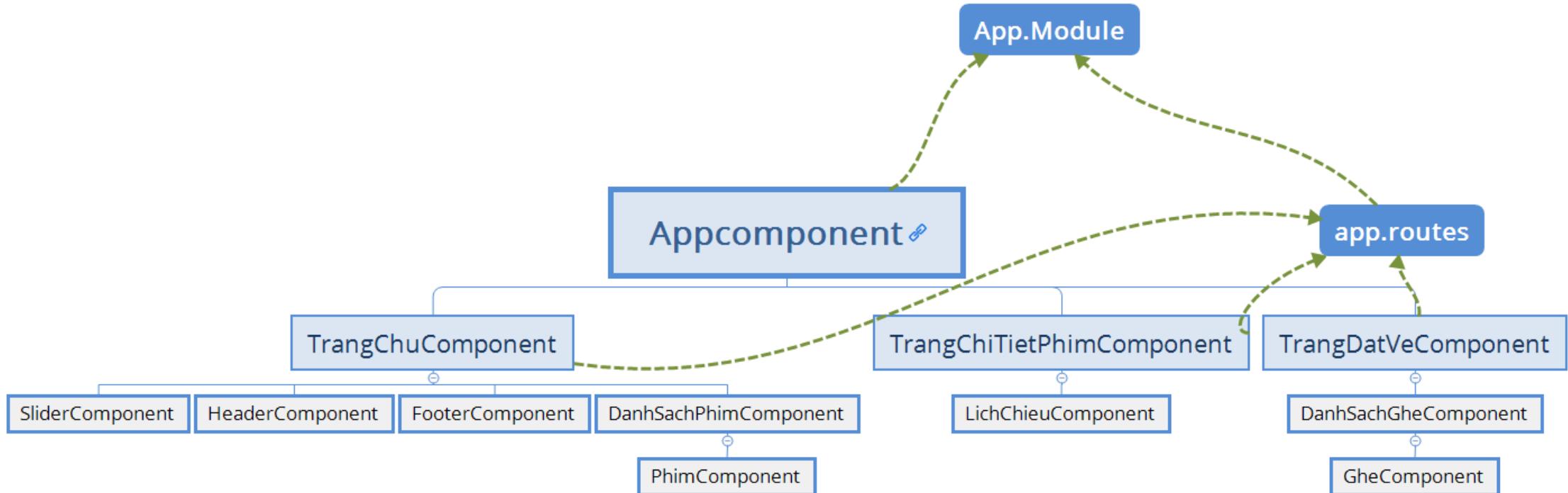
Trao đổi thảo luận (Hỏi + Trả lời)

Làm đồ án cuối khóa (hỏi đáp + google)

Giới thiệu về angular 6

1. **Angular js 2(4-5-6)** hay gọi tắt là **angular 2(4-5-6)** là 1 framework của javascript được google phát triển để phát triển ứng dụng web.
2. Angular **2(4-5-6)** là 1 phiên bản tiếp theo của angular js (angular 1) có thêm 1 số khái niệm mới. (Nếu biết angular 1 thì học **angular 2(4-5-6)** sẽ nhanh hơn tuy nhiên không biết cũng không sao chỉ cần nắm vững javascript là được).
3. **Angular 2(4-5-6)** hỗ trợ đa nền tảng web lẫn di động (dùng Ionic build ra các ứng dụng di động).
4. Viết chủ yếu bằng code typescript (gần với oop hơn dễ tiếp cận).
5. Hướng tiếp cận dựa trên component, module.

Kiến trúc của angular 2 (4-5-6)



❖ Module

App component (Thuộc app module)

Trang Chủ

COMPONENT

Trang Sản phẩm

COMPONENT



Để chuyển đổi
qua lại giữa các
trang (hay
component) ta có
khái niệm
RouteConfig

❖ Component

Component Trang chủ

Header component

Navigation
component

Content Component

Footer Component

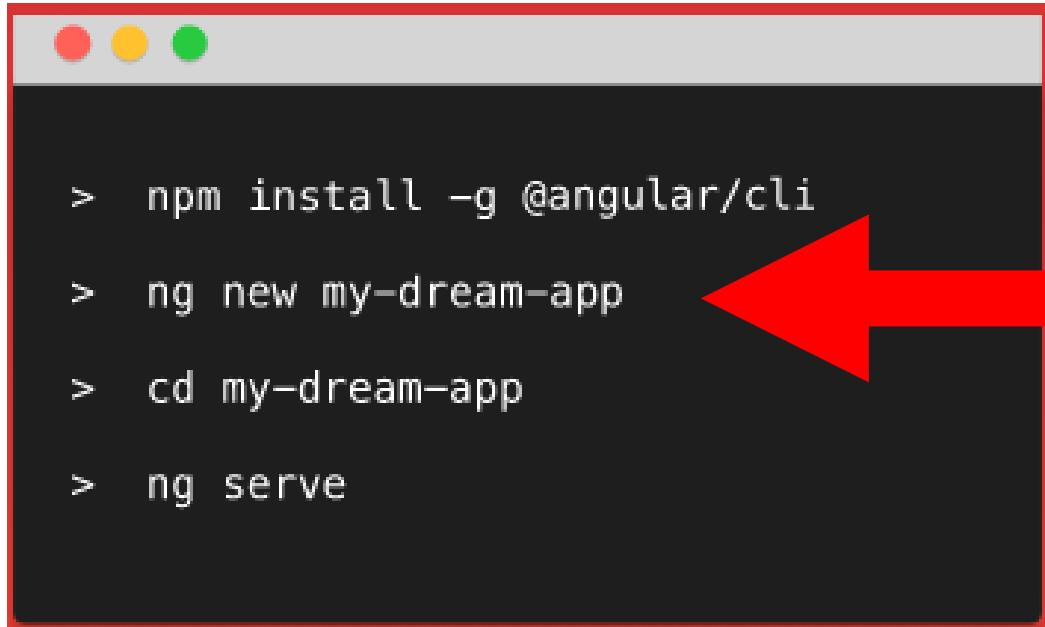
Một component được
cấu tạo từ các
component con và tất
cả component đó được
quản lý bởi 1 module.

Một số khái niệm trong angular 6

- Module
- Components
- Phân tách module (Nhóm component)
- Templates & Styles
- Metadata
- Databinding
- BindingCustomer
- Directives
- Pipes
- Input, Output, ViewChild
- Form-Validation
- Service - Observable – Object – HTTP
- Routing
- Parameter
- Get Post Put Delete (http service)
- Lưu trữ cục bộ - Redirect
- Guard
- NgRx
- Animation

Cài đặt angular CLI

1. Angular CLI là bộ công cụ do google cung cấp để phát triển ứng dụng bằng angular 2.
2. Angular Cli giúp tạo sẵn các cấu trúc thư mục ứng dụng và các tập tin cấu hình.



```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

**Cú pháp để tạo thư mục cho ứng dụng
my-dream-app (Tên ứng dụng)**

Trang angular: <https://angular.io/> (basic)

Trang aglcli: <https://cli.angular.io/>

Tài liệu: <https://github.com/angular/angular-cli/wiki>

Giới thiệu về cấu trúc thư mục và tập tin

The screenshot shows a code editor with a dark theme. At the top, there is a tab labeled 'package.json'. Below it is the JSON content of the file:

```
1 {  
2   "name": "phim",  
3   "version": "0.0.0",  
4   "license": "MIT",  
5   "scripts": {  
6     "ng": "ng",  
7     "start": "ng serve",  
8     "build": "ng build",  
9     "test": "ng test",  
10    "lint": "ng lint",  
11    "e2e": "ng e2e"  
12  },  
13  "private": true,  
14  "dependencies": {  
15    "@angular/animations": "^4.2.4"  
16  }  
17}  
18  
19
```

Below the code editor, there are several tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, showing the output of a webpack compilation:

```
chunk {polyfills} polyf  
chunk {styles} styles.b  
chunk {vendor} vendor.b  
  
webpack: Compiled successfully
```

Tương tự typescript thư mục package.json lưu trữ các thông tin của ứng dụng.

Bao gồm cả những gói tin thư viện được cài sẵn trong ứng dụng.

Trong đó chứa rất nhiều gói của angular 2 được cài sẵn

{} tsconfig.app.json ✘

```
1  {
2      "extends": "../tsconfig.json",
3      "compilerOptions": {
4          "outDir": "../out-tsc/app",
5          "baseUrl": "./",
6          "module": "es2015",
7          "types": []
8      },
9      "exclude": [
10         "test.ts",
11         "**/*.spec.ts"
12     ]
13 }
14
```

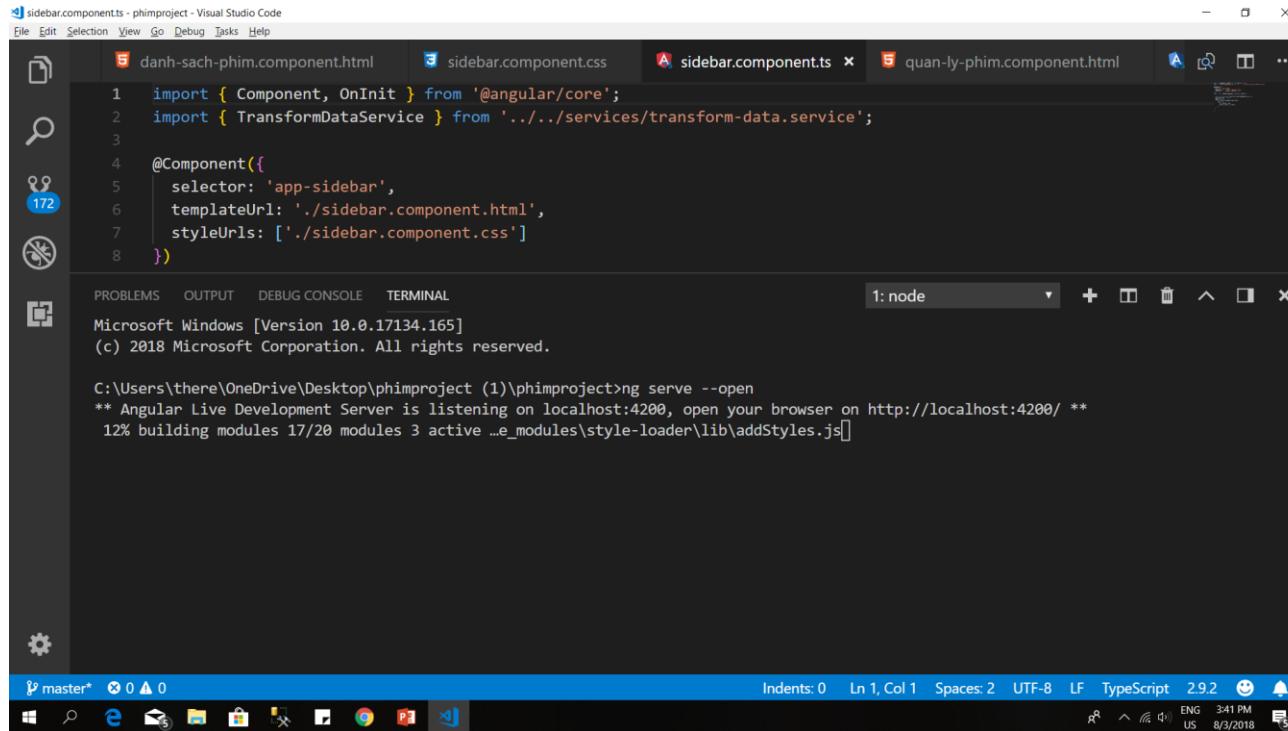
File này dùng để cấu hình biên dịch cú pháp typescript => js

```
1  [
2   "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
3   "version": 1,
4   "newProjectRoot": "projects",
5   "projects": {
6     "demoProject": {
7       "root": "",
8       "sourceRoot": "src", // Project type.
9       "projectType": "application",
10      "prefix": "app",
11      "schematics": {},
12      "architect": {
13        "build": {
14          "builder": "@angular-devkit/build-angular:browser",
15          "options": {
16            "outputPath": "dist/demoProject",
17            "index": "src/index.html",
18            "main": "src/main.ts",
19            "polyfills": "src/polyfills.ts",
20            "tsConfig": "src/tsconfig.app.json",
21            "assets": [
22              "src/favicon.ico",
23              "src/assets"
24            ],
25            "styles": [
26              "src/styles.css"
27            ],
28            "scripts": [
29            ]
30          }
31        }
32      }
33    }
34  }
35 }
```

File angular.json là file chính nó quy định toàn bộ cấu hình của app.

Cài đặt chạy thử ứng dụng angular đầu tiên

Sau khi tải và cài đặt thành công angular cli ta sử dụng cú pháp **ng serve -open** để project bắt đầu build và đóng gói tất cả các thư viện + source code (module component...)



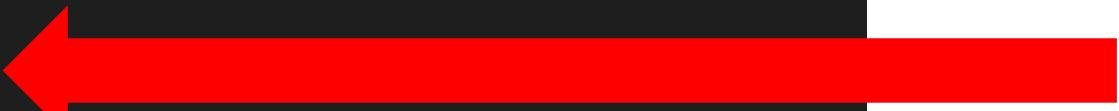
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `danh-sach-phim.component.html`, `sidebar.component.css`, `sidebar.component.ts`, and `quan-ly-phim.component.html`.
- Code Editor:** Displays the `sidebar.component.ts` file content:1 import { Component, OnInit } from '@angular/core';
2 import { TransformDataService } from '../../../../../services/transform-data.service';
3
4 @Component({
5 selector: 'app-sidebar',
6 templateUrl: './sidebar.component.html',
7 styleUrls: ['./sidebar.component.css']
8 })
- Terminal:** Shows the command `ng serve --open` being run, resulting in the message: "Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/". It also shows progress: "12% building modules 17/20 modules 3 active ...e_modules/style-loader/lib/addStyles.js".
- Status Bar:** Shows the current branch is `master`, and the system status includes Indents: 0, Ln 1, Col 1, Spaces: 2, UTF-8, LF, TypeScript 2.9.2, ENG, US, 3:41 PM, 8/3/2018.

Nguyên tắc hoạt động của project angular cli

Khi khởi động website nó sẽ chạy file **index.html** đầu tiên

Angular tự động đóng gói file và thêm vào các file cần thiết trong quá trình khởi chạy, trong đó có **main.ts**



```
index.html
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>Phim</title>
6    <base href="/">
7
8    <meta name="viewport" content="width=device-width, initial-scale=1">
9    <link rel="icon" type="image/x-icon" href="favicon.ico">
10   </head>
11   <body>
12     <app-root></app-root>
13   </body>
14 </html>
```

Component `<app-root>` chứa tất cả nội dung của toàn ứng dụng.

Kết nối main.ts sẽ gọi đến file app.module.ts

```
ts main.ts    ×

1 import { enableProdMode } from '@angular/core';
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5 import { environment } from './environments/environment';
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 //Câu lệnh dùng để khởi tạo appmodule để nó chạy đầu tiên
12 platformBrowserDynamic().bootstrapModule(AppModule)
13   .catch(err => console.log(err));
14
```

Dòng này khai báo cho angular biết
được AppModule là module gốc
quản lý toàn bộ ứng dụng web



app.module.ts



```
8  @NgModule({
9    declarations: [
10      AppComponent
11    ],
12    imports: [
13      BrowserModule, HomeModule, HttpModule, AdminModule
14    ],
15    providers: [],
16  }
17  bootstrap: [AppComponent]
18})
19
20 export class AppModule { }
```

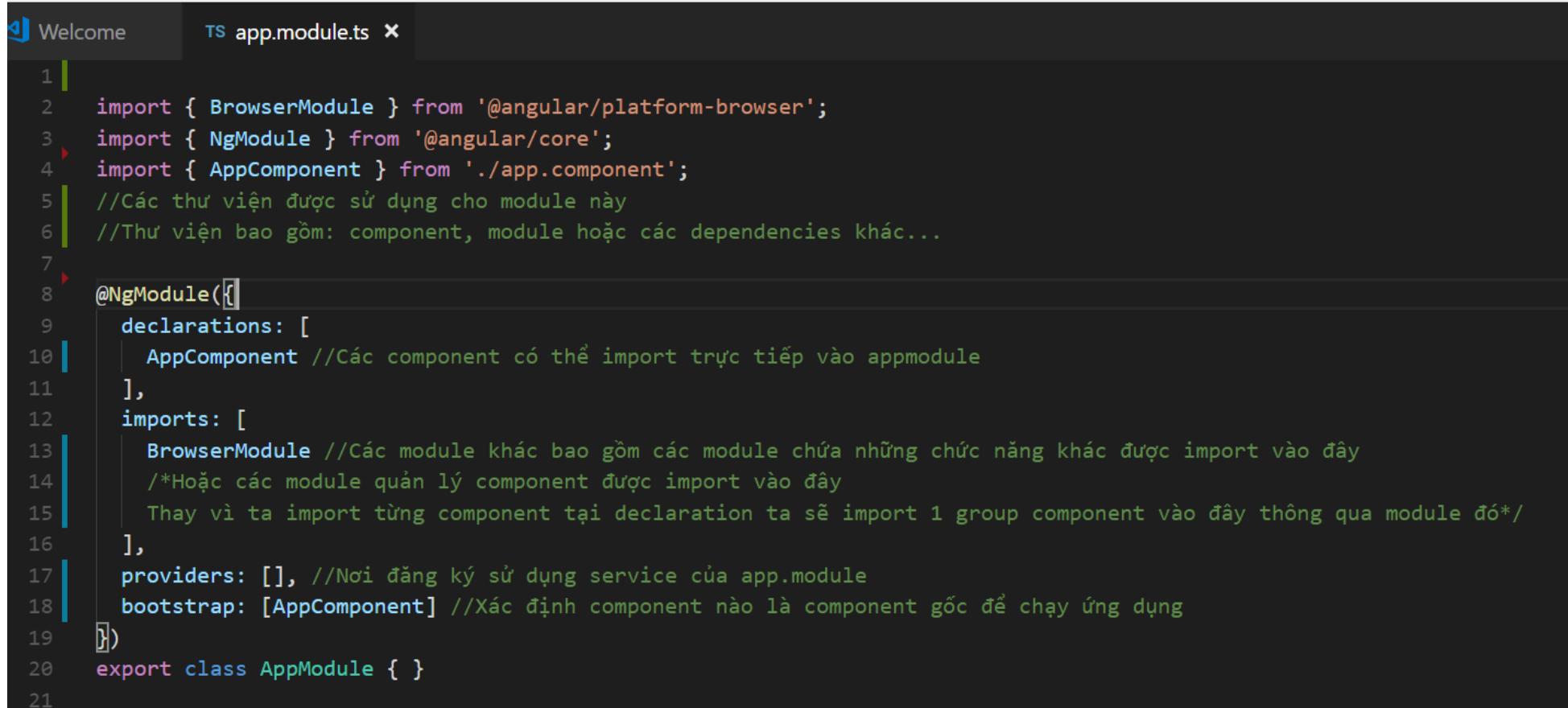


Nói cho angular biết được AppComponent là component gốc, cứ vào lấy html của nó parse ra là được

1. Khái niệm về module

- *Module là 1 Class dùng để đóng gói 1 chức năng cụ thể của ứng dụng.*
- *Có nhiều loại module ví dụ như:*
 - ❖ Browser module: được sử dụng ở trên chứa tất cả các **dependencies**(các gói thư viện từ *node_module* sử dụng cho ứng dụng) cần thiết để chạy Angular 2 trên trình duyệt.
 - ❖ HttpModule, FormModule, RoutingModule... (Ta sẽ tìm hiểu ở các phần sau).
- *Nếu xét về mối quan hệ giữa module và component thì module giống như 1 group của component quản lý các component. 1 Module có thể quản lý nhiều component và mỗi component phải được quản lý bởi module nào đó.*
- Đối với **app.module** thì đây là nơi bắt đầu khởi chạy của ứng dụng. Nó gọi là module gốc và nó **chứa 1 component gốc** là **app.component**.

Lấy ví dụ về app.module.ts (module chính trên toàn ứng dụng)



```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { AppComponent } from './app.component';
4 //Các thư viện được sử dụng cho module này
5 //Thư viện bao gồm: component, module hoặc các dependencies khác...
6
7 @NgModule({
8   declarations: [
9     AppComponent //Các component có thể import trực tiếp vào appmodule
10    ],
11   imports: [
12     BrowserModule //Các module khác bao gồm các module chứa những chức năng khác được import vào đây
13     /*Hoặc các module quản lý component được import vào đây
14     Thay vì ta import từng component tại declaration ta sẽ import 1 group component vào đây thông qua module đó*/
15    ],
16   providers: [], //Nơi đăng ký sử dụng service của app.module
17   bootstrap: [AppComponent] //Xác định component nào là component gốc để chạy ứng dụng
18 })
19 export class AppModule { }
```

Cú pháp tạo module

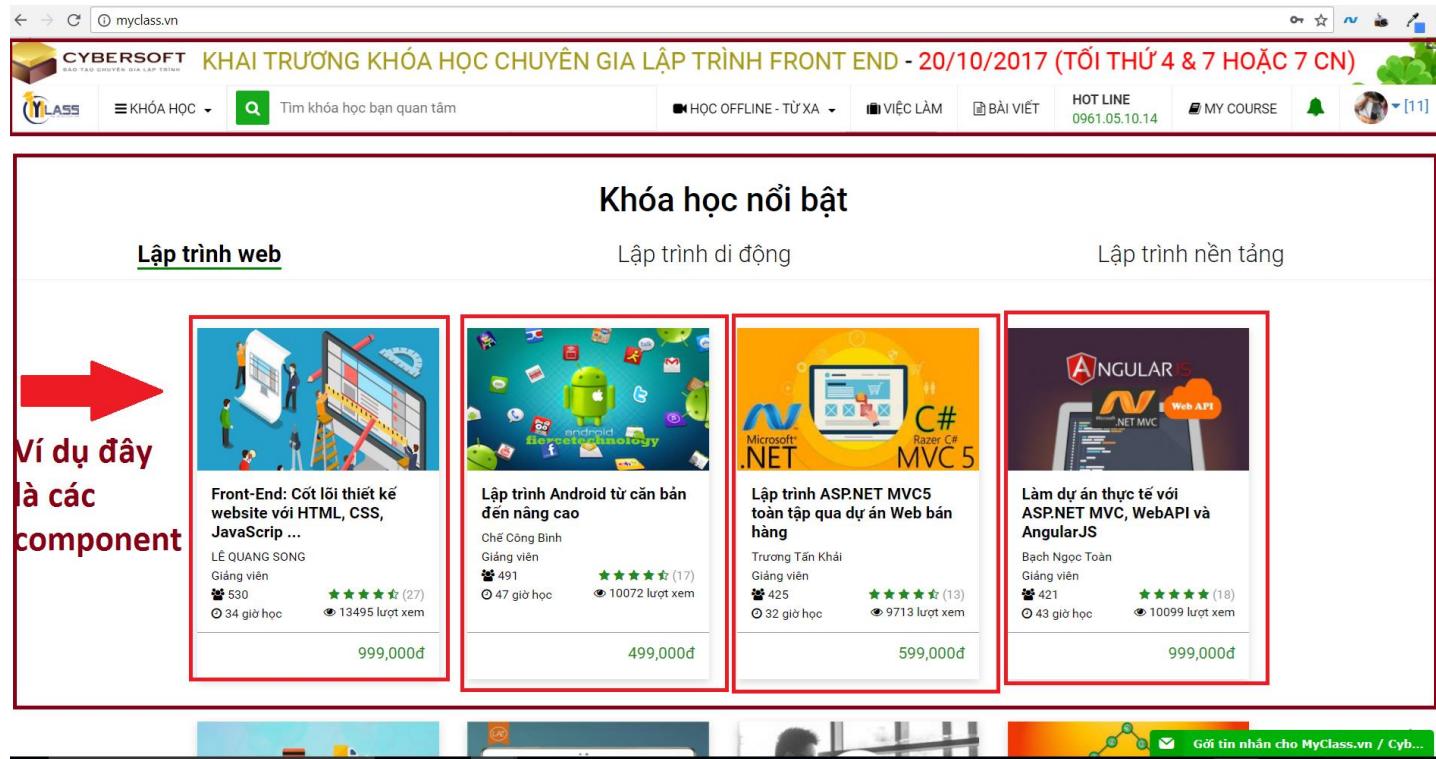
ng g module [tên module]

2. Khái niệm về component

- Component biểu diễn giao diện UI (file.html).
- Nói 1 cách đơn giản 1 component là 1 thẻ do mình định nghĩa trong thẻ đó chứa các nội dung html do mình biên soạn.
- Một component bao gồm:
 - + Giao diện html
 - + Css của giao diện html đó
 - + Selector (tên thẻ component do ta tự đặt)
 - + Ngoài ra còn chứa 1 class javascript để xử lý cho component đó và được export ra ngoài.

Ví dụ ta có ứng dụng web myclass.vn là app.module thì ta sẽ có component chứa tất cả các trang khác là app.component.

- + Trong app.component chứa TrangChuComponent.
 - + Rồi trong trangchu lại chứa các thành phần khóa học component,
- + Trong app.component cũng chứa Trang chi tiết khóa học component
 - + Rồi trong trang chi tiết cũng lại chứa các component khóa học và 1 số component khác



Ví dụ: Ta thử tạo 1 component header tại appmodule.

Trong header chứa các file định nghĩa header component

The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure under 'STARTANGULAR2_5'. It includes 'src' with 'app' and 'header' folders. Inside 'header', there are files: '# header.component.css', 'header.component.html', 'TS header.component.spec.ts', and 'TS header.component.ts'. The 'header.component.ts' file is open in the editor. A yellow box highlights the code block that defines the @Component decorator:

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-header',
5   templateUrl: './header.component.html',
6   styleUrls: ['./header.component.css']
7 })
8 export class HeaderComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit() {
13   }
14 }
15
16
```

Tiến hành đăng ký cho app module quản lý HeaderComponent

The screenshot shows the 'app.module.ts' file in the 'Welcome' tab. A yellow box highlights the import statement for HeaderComponent:

```
1
2 import { BrowserModule } from '@angular/platform-browser';
3 import { NgModule } from '@angular/core';
4 import { AppComponent } from './app.component';
5 //Import component header vào app.module
6 import { HeaderComponent } from './header/header.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent, HeaderComponent
11   ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
```

Cú pháp tạo component

ng g component [tên component]

Để hiển thị HeaderComponent ta làm thế nào ???

EXPLORER

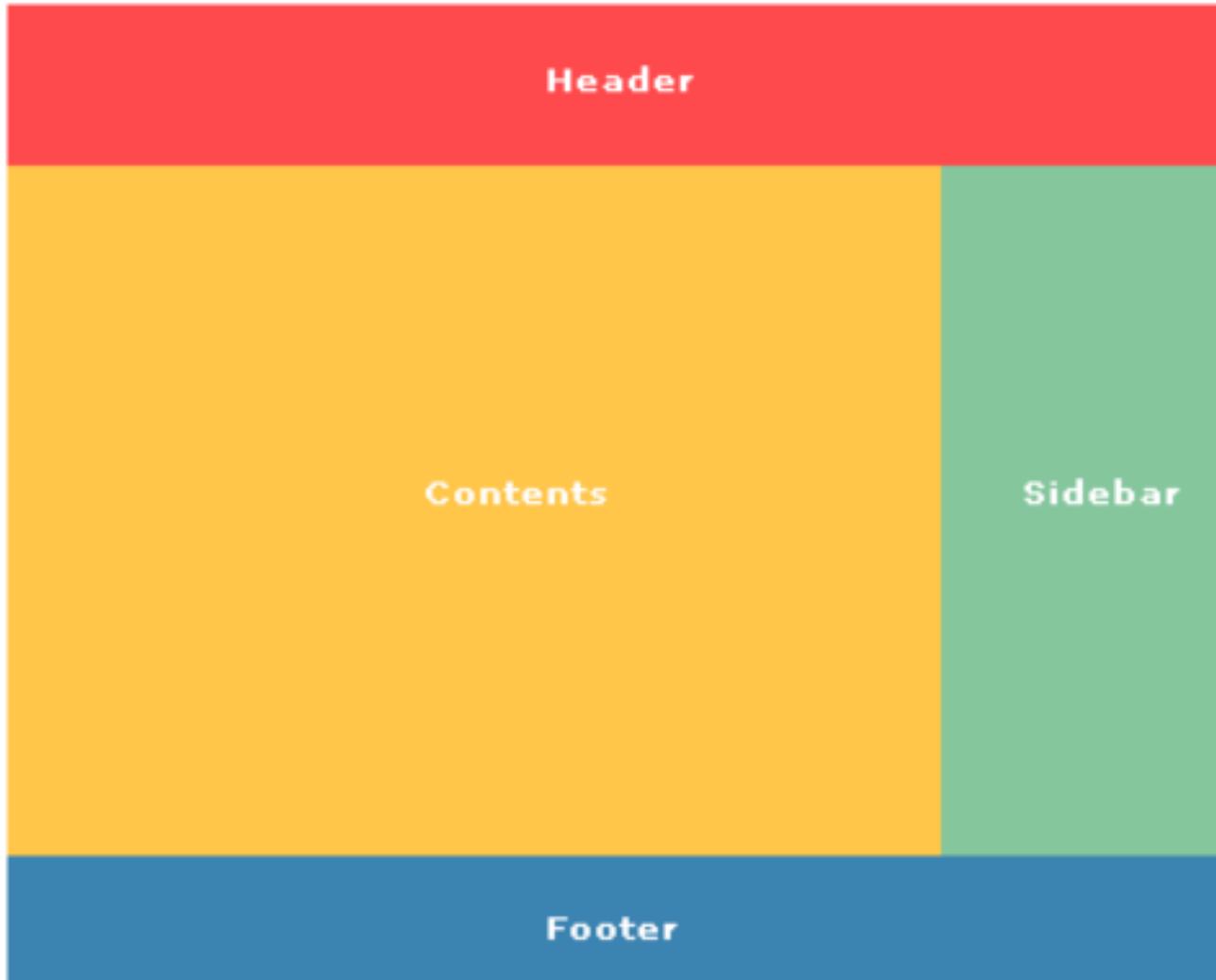
OPEN EDITORS

STARTANGULAR2_5

- e2e
- node_modules
- src
 - app
 - header
 - # header.component.css
 - header.component.html
 - TS header.component.spec.ts
 - TS header.component.ts
- # app.component.css
- app.component.html M
- TS app.component.spec.ts
- TS app.component.ts
- TS app.module.ts M

```
app.component.html TS header.component.ts x
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-header',
5   templateUrl: './header.component.html',
6   styleUrls: ['./header.component.css']
7 })
8 export class HeaderComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit() {
13   }
14 }
15 }
```

Bài tập:



Yêu cầu:

- Chia component như hình đưa vào app.component.
- Gợi ý:
 - ✓ Sử dụng bootstrap bằng cách chèn cdn vào file index.html.
 - ✓ Sau đó chia layout (Chia cột) rồi chèn các thẻ component vào các cột.

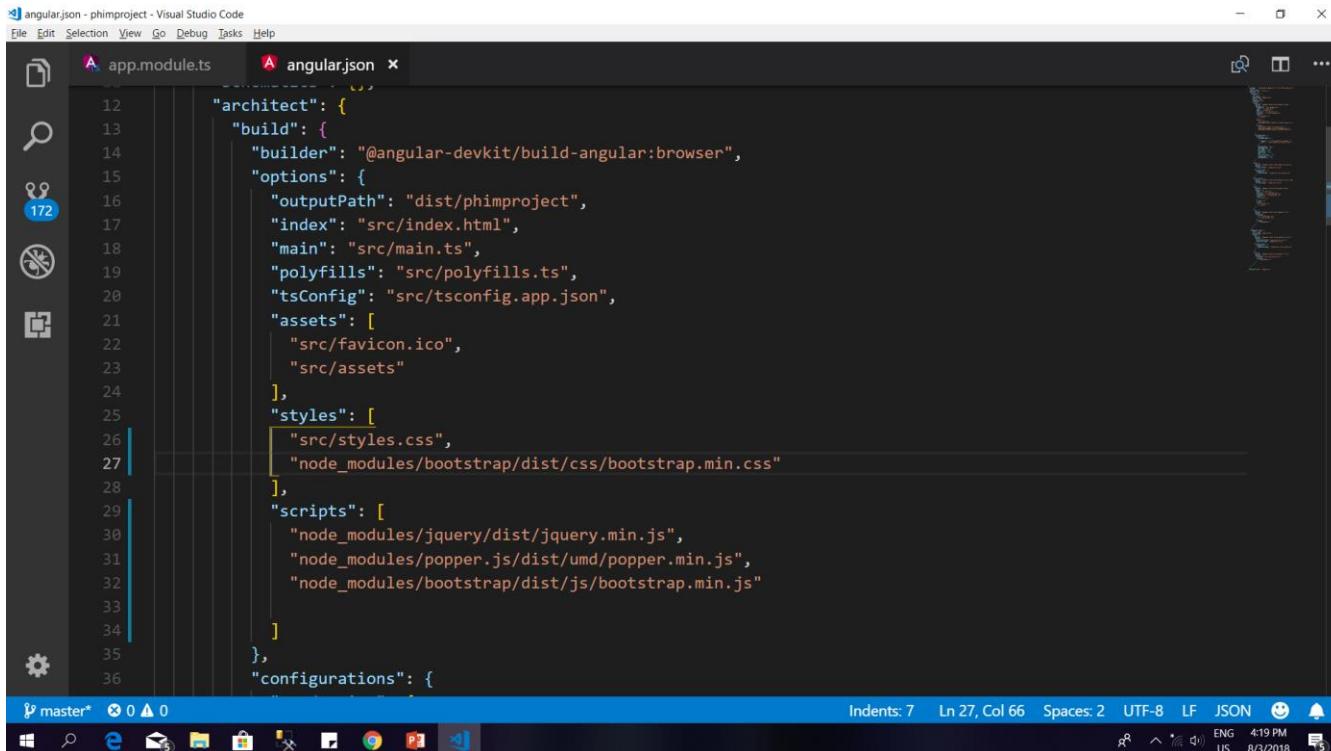
Hướng dẫn cài đặt bootstrap 4

❖ Cách 1:

Đưa đường dẫn bootstrap vào file index.html (Cách dễ nhất)

❖ Cách 2:

1. npm install bootstrap jquery popper.js --save
2. Vào file **angular.json** thêm vào đường dẫn tới css và javascript để đóng gói webpack.
4. Tiến hành **ng serve** lại



```
angular.json - phimproject - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
app.module.ts angular.json
12     "architect": {
13         "build": {
14             "builder": "@angular-devkit/build-angular:browser",
15             "options": {
16                 "outputPath": "dist/phimproject",
17                 "index": "src/index.html",
18                 "main": "src/main.ts",
19                 "polyfills": "src/polyfills.ts",
20                 "tsConfig": "src/tsconfig.app.json",
21                 "assets": [
22                     "src/favicon.ico",
23                     "src/assets"
24                 ],
25                 "styles": [
26                     "src/styles.css",
27                     "node_modules/bootstrap/dist/css/bootstrap.min.css"
28                 ],
29                 "scripts": [
30                     "node_modules/jquery/dist/jquery.min.js",
31                     "node_modules/popper.js/dist/umd/popper.min.js",
32                     "node_modules/bootstrap/dist/js/bootstrap.min.js"
33                 ]
34             },
35             "configurations": {
36             }
37         }
38     }
39 }
```

Indents: 7 Ln 27, Col 66 Spaces: 2 UTF-8 LF JSON ⚡ 🌐

8/3/2018 4:19 PM US ENG

Bài tập :

The diagram illustrates a website layout structure with the following components:

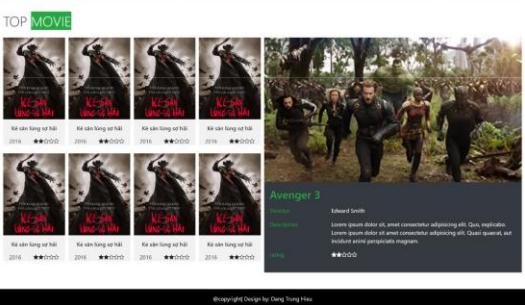
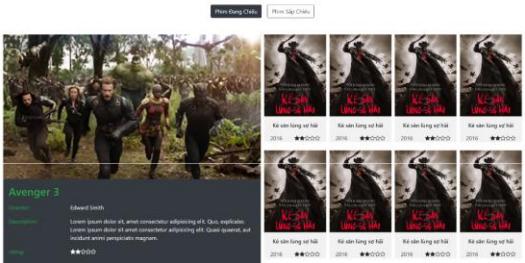
- headercomponent**: A dark header bar at the top with a yellow border containing navigation links: Home, About, Services, Contact.
- IndexComponent (viền vàng)**: A large yellow section at the top containing:
 - Business Name or Tagline: "1920 x 400"
 - SliderComponent (Viền tím): A purple rectangular area.
- What We Do**: A section with a grey background containing text and a "Call to Action" button.
- Contact Us**: A section with a green background containing contact information.
- IndexContentComponent (Viền lục)**: A green section containing three cards, each with a "Card title" and a "Find Out More" button.
- ItemComponent (Lưu ý thiết kế 1 component sau đó copy ra 3 component)**: A purple section containing three cards, each with a "Card title" and a "Find Out More" button.
- FooterComponent**: A dark footer bar at the bottom with a red border containing copyright information: "Copyright © Your Website 2017".

Yêu cầu:

- + Tạo và thiết kế component tương ứng.
- + Bố cục giao diện các component như hình ảnh.

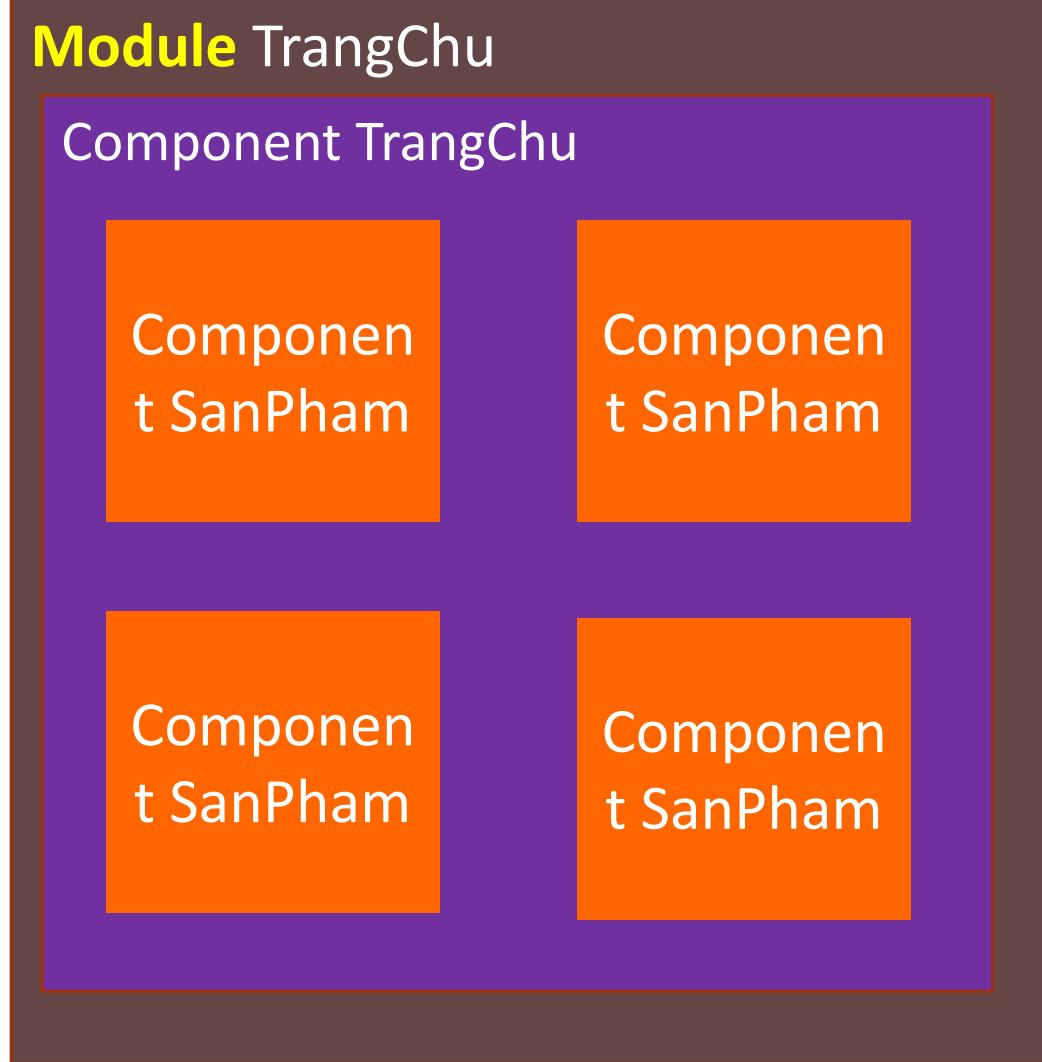
Lưu ý: Không cần dàn layout chi tiết cho nội dung từng component, có thể copy bootstrap

Bài tập dàn layout có nội dung:



- Yêu cầu phân tích xem trang này có bao nhiêu component.
- Xây dựng các component tương ứng.

3. Phân tách module – Nhóm các component



- ❖ Module là gì ?
 - ❖ Tại sao phải phân tách module?
 - ❖ Phân tách module như thế nào ?
- ✓ **Module** là 1 **nhóm chứa** các **component** để **thực hiện** 1 **chức năng** nào đó của ứng dụng.
- ✓ Việc phân tách module giúp ta **dễ quản lý** ứng dụng, **chia nhỏ công việc**, **dễ dàng bảo trì** và **nâng cấp**.
- ✓ **Phân tách module** dựa trên **nhóm** các **component** **phục vụ** cho 1 **chức năng** nào đó.
- ✓ **Ví dụ:** Nhìn vào hình bên ta thấy **Module Trang chủ** chứa **các component** để **hình thành** **nên trang chủ**.

AppModule

AppComponent

Module Home

Component TrangChu

Component
Slider

Component
header

Component
Content

Component
Footer ...

Module AdminQLSP

Component Them, Sua, XemChiTiet

Component
ThemSP

Component
SuaSP

Component
XemChiTiet

Component
...

Lấy ví dụ về Layout module (Một module con)

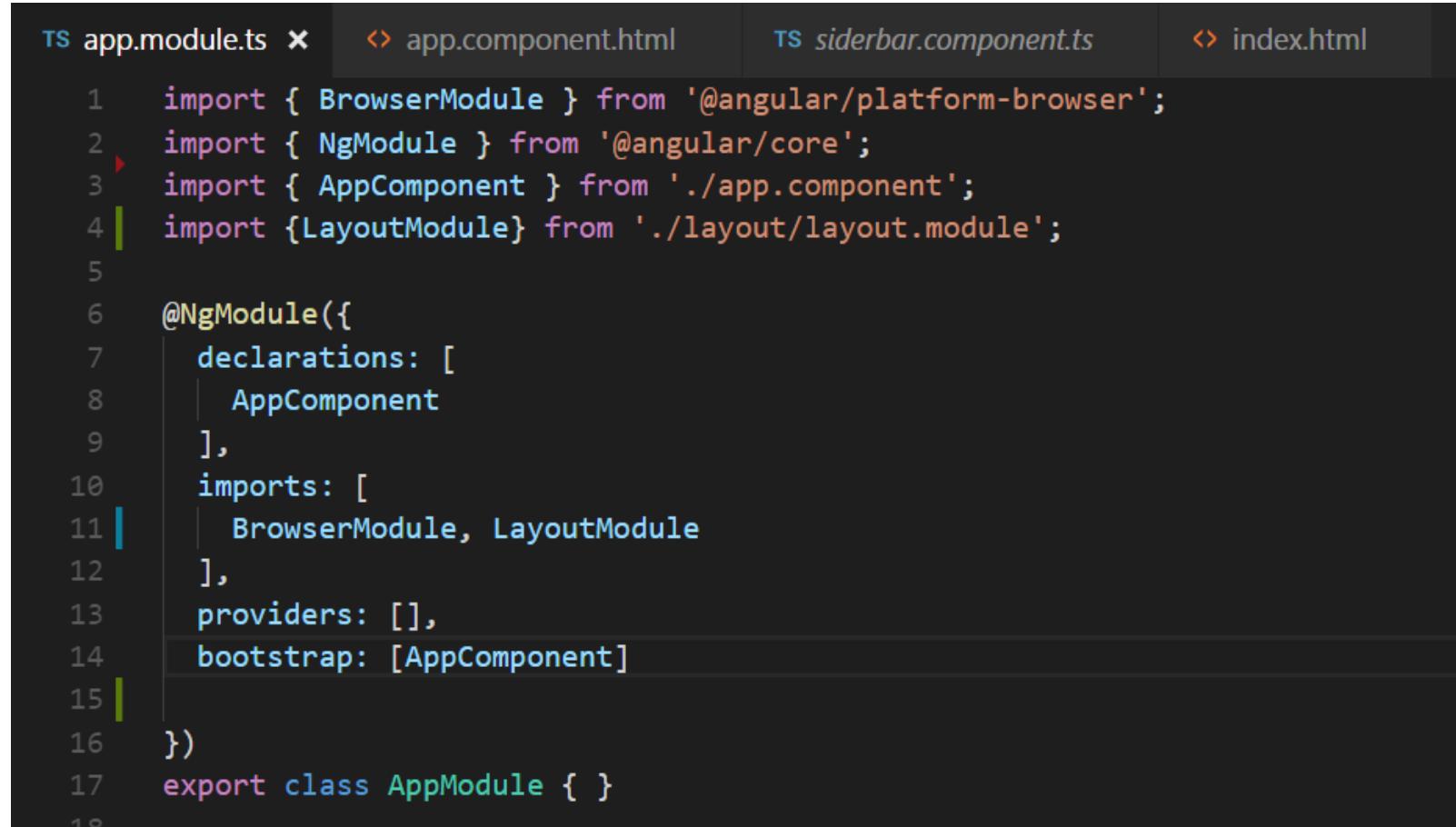
```
ts app.module.ts      ts layout.module.ts x
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { HeaderComponent } from './header/header.component';
4 import { SidebarComponent } from './sidebar/sidebar.component';
5 import { FooterComponent } from './footer/footer.component';
6
7 @NgModule({
8   imports: [
9     CommonModule
10 ],
11   exports:[HeaderComponent,SidebarComponent,FooterComponent],
12   //Exports những component của module này ra bên ngoài để
13   //khi ở module khác import module này vào có thể sử dụng được những component này
14   declarations: [HeaderComponent, SidebarComponent, FooterComponent]
15   //declarations: các component được import vào module này (giống như khai báo component)
16 })
17 export class LayoutModule {}
```

Ví dụ ta có 1 module **layout** chứa các component: **Header**, **Footer**, **Sidebar**

Câu hỏi đặt ra vậy làm cách nào ta đưa được các component Header, Footer, SiderBar vào app.component sử dụng ???

→ Ta phải **exports** các **component** đó ra thì khi **import module** này vào **app.module** component này mới được hiểu

- Tại app.module thay vì muốn dùng thì phải khai báo ở declaration từng component nhưng do ta gom nhóm module lại nên chỉ cần imports module đó vào là được



```
TS app.module.ts ✘    < app.component.html      TS sidebar.component.ts      < index.html

1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { AppComponent } from './app.component';
4 import {LayoutModule} from './layout/layout.module';
5
6 @NgModule({
7   declarations: [
8     AppComponent
9   ],
10  imports: [
11    BrowserModule, LayoutModule
12  ],
13  providers: [],
14  bootstrap: [AppComponent]
15 })
16 )
17 export class AppModule { }
```

3. Khái niệm về Template và Styles

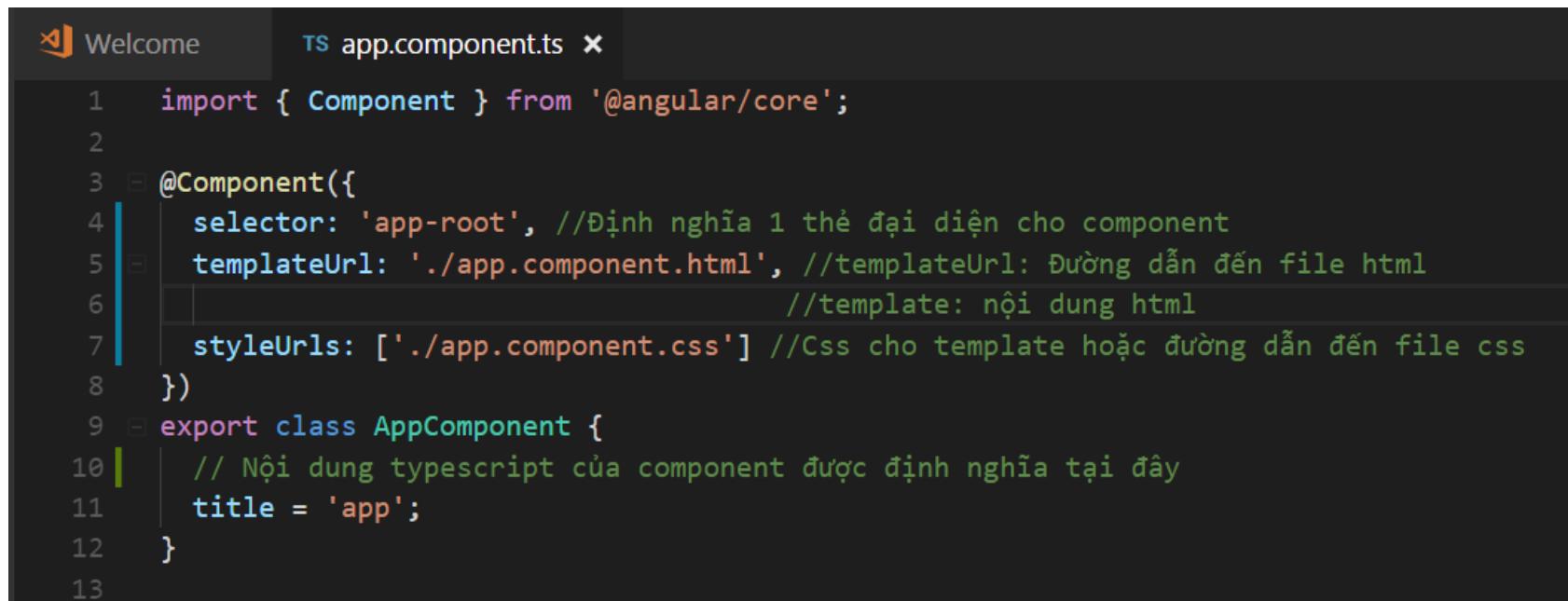
Template là 1 thành phần html của component.

Trong 1 component có 2 dạng định nghĩa template.

+**Template**: Định nghĩa nội dung html của component đó trực tiếp

+**TemplateUrl**: Định nghĩa nội dung html của component đó thông qua url đến file .html.

Tương tự style cũng có **Styles**, và **StyleUrls**



The screenshot shows the VS Code interface with the 'app.component.ts' file open. The code defines an Angular component with the following properties:

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root', //Định nghĩa 1 thẻ đại diện cho component
5   templateUrl: './app.component.html', //templateUrl: Đường dẫn đến file html
6   //template: nội dung html
7   styleUrls: ['./app.component.css'] //Css cho template hoặc đường dẫn đến file css
8 })
9 export class AppComponent {
10   // Nội dung typescript của component được định nghĩa tại đây
11   title = 'app';
12 }
13
```

Annotations in the code explain the purpose of each property:

- `selector: 'app-root'` is annotated as "Định nghĩa 1 thẻ đại diện cho component".
- `templateUrl: './app.component.html'` is annotated as "templateUrl: Đường dẫn đến file html" and "template: nội dung html".
- `styleUrls: ['./app.component.css']` is annotated as "Css cho template hoặc đường dẫn đến file css".
- `// Nội dung typescript của component được định nghĩa tại đây` is annotated as "Nội dung typescript của component được định nghĩa tại đây".

4. Metadata

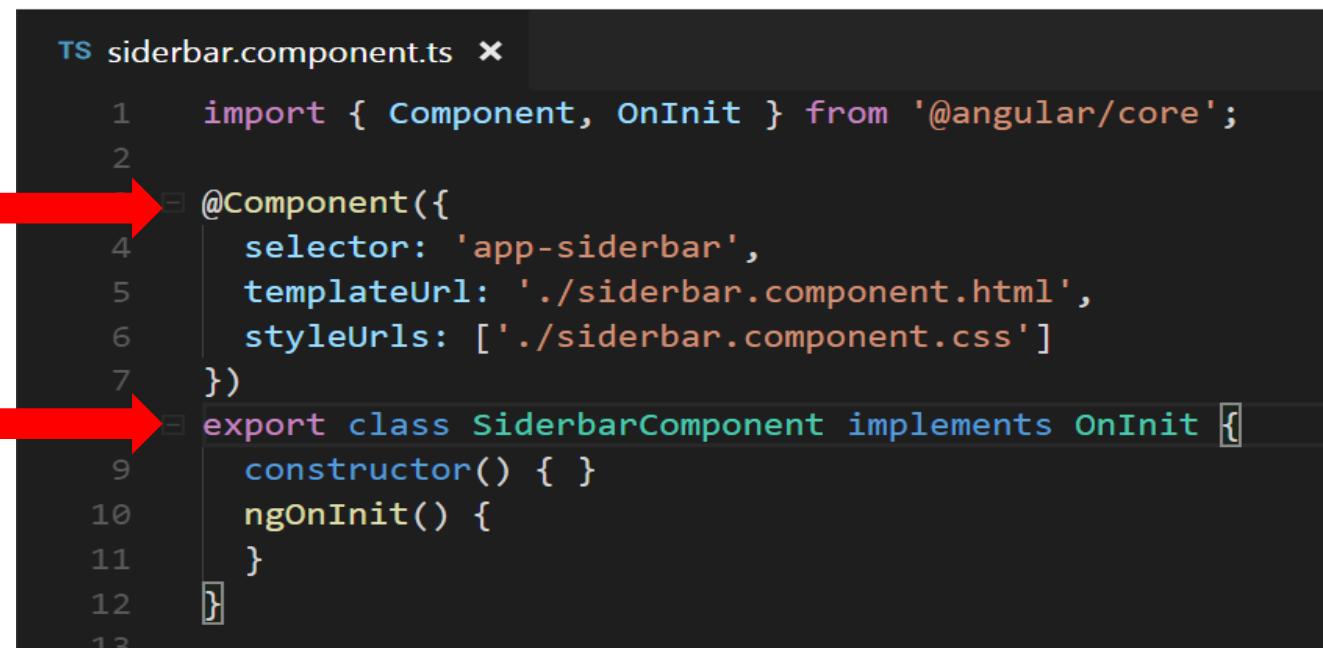
Thực chất 1 **component** chỉ là 1 **class**. Nó không phải là 1 component cho tới khi ta khai báo nó với **angular**.

Để khai báo 1 **class** với **angular** rằng nó là 1 **component**, ta sẽ gắn thẻ **metadata** vào class này.

Trong **typescript** để gắn thẻ metadata ta dùng **decorator**

Ở đây **@Component** chính là **decorator**

Class **SiderbarComponent** được định nghĩa thành component khi có **@Component**



```
TS siderbar.component.ts ×
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-siderbar',
5   templateUrl: './siderbar.component.html',
6   styleUrls: ['./siderbar.component.css']
7 })
8 export class SiderbarComponent implements OnInit {
9   constructor() { }
10  ngOnInit() {
11    }
12 }
13
```

5. Databinding

- Data binding là 1 tính năng 1 đặc tính của framework hiện đại chúng đồng bộ dữ liệu được hiển thị trên view(template html) và model (class typescript).
- Nói 1 cách đơn giản databinding là sự giao tiếp giữa Typescript và HTML

❑ Các cơ chế binding dữ liệu angular 2

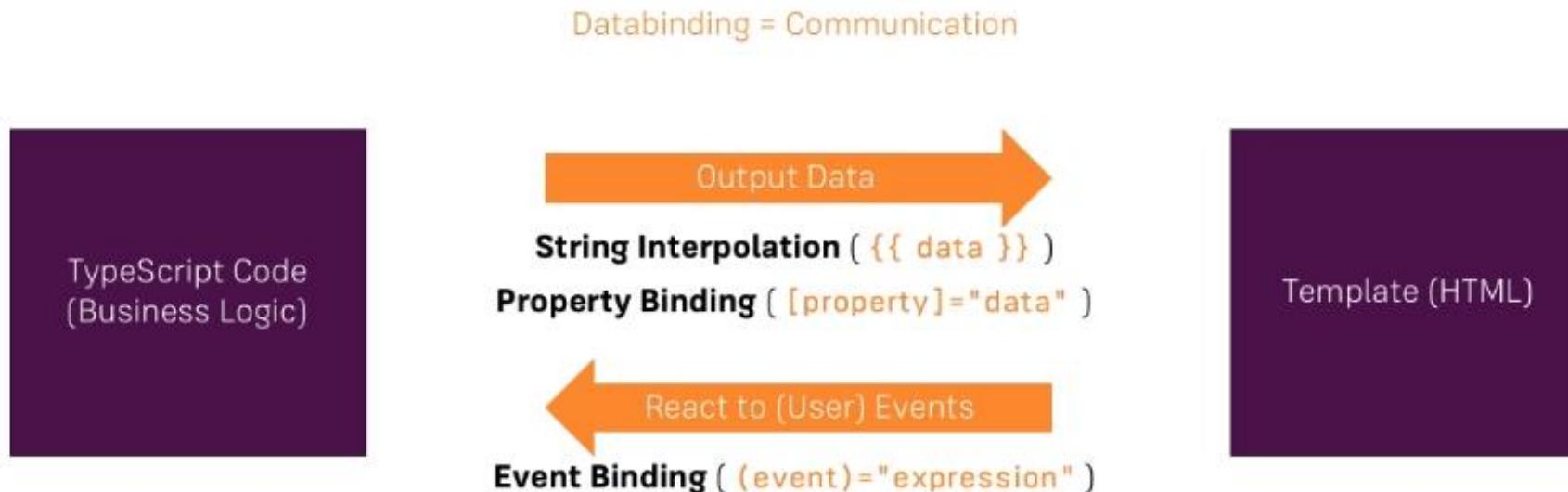
❖ One-waybinding

- Interpolation
- Property binding
- Event binding

❖ Two-waybinding

❖ Oneway binding

Understanding Databinding



❖ Oneway binding (Interpolation)

```
ts onewaybinding.component.ts x
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-onewaybinding',
5   template: `<input value="{{name}}>` ,
6   styleUrls: ['./onewaybinding.component.css']
7 })
8 export class OnewaybindingComponent implements OnInit {
9   name:string = "cybersoft";
10  constructor() { }
11  ngOnInit() {
12  }
13 }
14
```

Thế nào là one-way binding **one-way binding** sử dụng các biến trong class **component** binding ra view hay còn gọi là template (**Interpolation Binding**) hoặc ngược lại thông qua sự kiện (**eventbinding**).

Interpolation Binding được viết với cú pháp **{{tenbien}}** hoặc **{{tenham()}}**. Nếu dữ liệu model thay đổi => view thay đổi.

❖ Oneway binding (Property binding)

Property binding cũng là một dạng one-way binding dữ liệu từ model được binding ra template (view) dưới dạng các thuộc tính của thẻ kết hợp dấu ngoặc []

Ví dụ:

```
TS propertybinding.component.ts ×  
1 import { Component, OnInit } from '@angular/core';  
2  
3 @Component({  
4   selector: 'app-propertybinding',  
5   template: `<input [value] = 'name'>`, ←  
6   styleUrls: ['./propertybinding.component.css']  
7 })  
8  
9 export class PropertybindingComponent implements OnInit {  
10   public name:string = "cybersoft";  
11   constructor() { }  
12  
13   ngOnInit() {  
14   }  
15  
16 }  
17
```

Binding thông qua thuộc tính value của
thẻ input: **[value]**

❖ Oneway binding (Event Binding)

Event binding thì ngược lại với 2 cách binding trước dữ liệu được binding từ view về model thông qua các sự kiện (Giống với event trong javascript hay jQuery).

```
TS eventbinding.component.ts ×  
1 import { Component, OnInit } from '@angular/core';  
2 @Component({  
3   selector: 'app-eventbinding',  
4   template:  
5     <input type="text" />  
6     <button (click)="DisplayName()" >Submit</button>  
7   ,  
8   styleUrls: ['./eventbinding.component.css']  
9 })  
10 export class EventbindingComponent implements OnInit {  
11   public name:string = "cybersoft";  
12   constructor() { }  
13   DisplayName()  
14   {  
15     console.log(this.name);  
16   }  
17   ngOnInit() {  
18   }  
19  
20 }  
21
```

Để binding với sự kiện click ta dùng:
(sự kiện) = “phương thức xử lý”

Sự kiện ở đây là click, pt DisplayName
(click) = “DisplayName()”

❖ Event Binding(tt)

Event binding với tham số

```
ts eventbinding.component.ts x

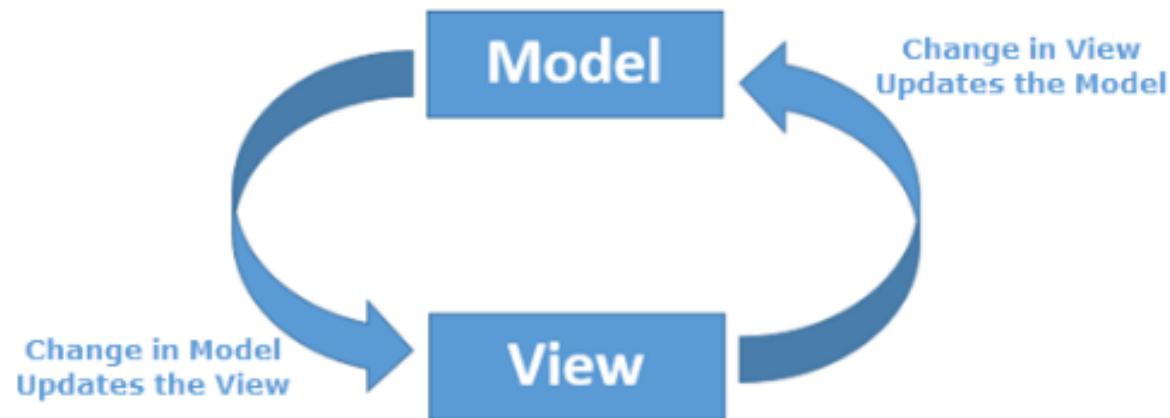
1 import { Component, OnInit } from '@angular/core';
2 @Component({
3   selector: 'app-eventbinding',
4   template: `
5     <input type="text" #thamso index='12' />
6     <button (click)="DisplayName(thamso)">Submit</button>
7   `,
8   styleUrls: ['./eventbinding.component.css']
9 })
10 export class EventbindingComponent implements OnInit {
11   public name:string = "cybersoft";
12   constructor() { }
13   DisplayName(thamso)
14   {
15     console.log(thamso.value); //Lấy value
16     //console.log(thamso.getAttribute("index")); //Lấy attribute
17   }
18   ngOnInit() {
19   }
20
21 }
22 }
```

#thamso đại diện cho thẻ input. Muốn lấy giá trị value của thẻ thì **.value** tương tự muốn lấy các thuộc tính khác thì **.thuộc tính khác**

❖ Two-way binding

Là một sự kết hợp giữa property binding và event binding.

- + Model thay đổi => view thay đổi
- + View thay đổi => model thay đổi



Sử dụng two way binding phải

import {FormsModule} from '@angular/forms'.

```
ts twowaybinding.component.ts ✘
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-two-way-binding',
5   template: `<input [(ngModel)] = "name"> {{name}}`,
6   styleUrls: ['./two-way-binding.component.css']
7 })
8 export class TwoWayBindingComponent implements OnInit {
9   public name:string = "cybersoft";
10  constructor() { }
11
12  ngOnInit() {
13  }
14
15 }
```

[(ngModel)] = ‘thuộc tính’

Là một sự kết hợp giữa event và property binding. ngModel tự nhận ra sự thay đổi của 1 trong 2 để tiến hành cập nhật cho phần còn lại.

Two-way binding thường áp dụng cho chức năng như thêm hay chỉnh sửa.

Bài tập:

Register form

Email:

Dùng [(ngModel)]

FullName:

Enter full name

Submit

Email: Dùng [(ngModel)]

FullName: Dùng event binding

Yêu cầu:

- + Tạo 1 component với nội dung như hình vẽ.
- + Dùng **interpolation** để **binding** 2 thuộc tính **Email** và **FullName** ra 2 dòng text dưới nút submit.
- + Dùng **event binding** để khi người dùng nhập vào **FullName** nhấn nút **submit** thì phần text màu cam thay đổi thành nội dung nhập vào
- + Dùng **two-way binding** để thực hiện khi người dùng gõ vào **Email** thì dòng text màu đỏ phía dưới thay đổi

6. Directive (Attribute directive)

Directives là một thành phần mở rộng HTML, hay nói cách khác là các thuộc tính (properties) của các thẻ HTML mà Angular nó định nghĩa thêm. Vì nó của riêng của Angular nên phải tuân thủ theo nguyên tắc của nó.

Có 3 loại directive:

- ❖ Components – directives
- ❖ Structural – directives
- ❖ Attribute – directives

❖ Structural – directives

Dùng để thay đổi diện mạo của dom ví dụ như ẩn hiện, load dữ liệu ...

- *ngIf
- *ngSwitch
- *ngFor

❖ Structural – directives

*ngIf: dùng để ẩn hiện theo điều kiện nào đó là true hoặc là false.

```
1 import { Component, OnInit } from '@angular/core';
2 @Component({
3   selector: 'app-directive',
4   template: `
5     <div *ngIf='status'>cybersoft</div>
6     <button (click) = "Hidden()"> Hidden </button>
7     <button (click) = "Show()"> Show </button>
8   `,
9   styleUrls: ['./directive.component.css']
10 })
11 export class DirectiveComponent implements OnInit {
12   public status:boolean = true;
13   Hidden()
14   {
15     this.status = false;
16   }
17   Show()
18   {
19     this.status = true;
20   }
}
```

Xem ví dụ ta thấy:

Ta có 1 div với nội dung cybersoft.

Div này được **hiển thị** khi giá trị **status = true**, và **ẩn đi** khi giá trị **status = false**.

Ngoài ra ta còn có thể thay thế biến đó thành 1 **biểu thức điều kiện**

Ví dụ:

<div *ngIf=(number>2)></div>

❖ Structural – directives

*ngIf else: Ở những phiên bản sau angular hỗ trợ ta thêm directive **ng-template** và **ngIf else**

- **ng-template:**

Về tính chất nó giống như 1 component (tái sử dụng) tuy nhiên phạm vi sử dụng chỉ được khai báo và sử dụng trong component chứa nó thôi.

- **Ng-IfElse:**

Giống như các ngôn ngữ lập trình directive if else giúp ta làm nhiệm vụ khi điều kiện if không thỏa (Thay vì phải dùng 2 lệnh if).

Ví dụ về ngIf else và ng-template:

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

@Component({
  selector: 'app-directive-if-else',
  template:
`<button (click)="DangNhap()">Đăng Xuất</button>
<div class="" *ngIf="status; else NoiDungElse">
  <h5>Xin chào <b>cybersoft</b></h5>
</div>
<ng-template #NoiDungElse>
  <h5>Chưa đăng nhập !</h5>
</ng-template>
,
styleUrls: ['./directive-if-else.component.css']
})
export class DirectiveIfElseComponent implements OnInit {
  public status:boolean = true;
  constructor() { }
  DangNhap(){
    this.status = false;
  }
  ngOnInit() {
  }
}
```



Khi điều kiện if không thỏa nó sẽ hiển thị phần nội dung của ng-template

Ý nghĩa:

+ Ban đầu ta có thuộc tính **status = true**.

+ Ta xét điều kiện **if** của biến **status** này nếu = **true** nó sẽ **hiển thị** phần nội dung phía trong là **xin chào cybersoft**.

+ Ngược lại (**else**) thì nó sẽ **hiển thị** phần ng-template có **#NoiDungElse** lên mà không hiển thị dòng xin chào cybersoft.

+ Để thử trường hợp ngược lại ta xây dựng 1 nút button với sự kiện click làm thay đổi trạng thái của biến status này.

❖ Structural – directives

*ngSwitch: dùng để ẩn hiện theo điều kiện giá trị điều kiện

```
2
3 @Component({
4   selector: 'app-ngswitch',
5   template: `
6     <div [ngSwitch]="dkSwitch" >
7       <div *ngSwitchCase="'red'" style="color:red"> Màu đỏ </div>
8       <div *ngSwitchCase="'blue'" style="color:blue"> Màu xanh </div>
9       <div *ngSwitchCase="'green'" style="color:green"> Màu xanh lá </div>
10      <div *ngSwitchDefault style="color:orange"> Màu cam mặc định </div>
11    </div>
12    <select #t (change)=>changeColor(t.value)>
13      <option value="red">red</option>
14      <option value="green">green</option>
15      <option value="blue">blue</option>
16      <option value="orange">orange</option>
17    </select>
18  `,
19  styleUrls: ['./ngswitch.component.css']
20 })
21 export class NgswitchComponent implements OnInit {
22   public dkSwitch:string = 'red';
23   changeColor(value){
24     this.dkSwitch = value;
25 }
```

[ngSwitch]=“dieukien”
tag nào (chứa giá trị
*ngSwitchCase = “giá trị
của biến điều kiện”
Thì tag đó được hiển thị

❖ Structural – directives

*ngFor: Cấu trúc lặp dùng để duyệt mảng hoặc danh sách các phần tử trong mảng object.

```
TS ng-for.component.ts ●

1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-ng-for',
5   template: `
6     <h1>Duyệt mảng object</h1>
7     <div *ngFor="let item of DSNV">{{item.ten}}: {{item.tuoi}}</div>
8     <h1>Duyệt mảng thông thường </h1>
9     <div *ngFor="let item of Colors">{{item}}</div>
10    `,
11   styleUrls: ['./ng-for.component.css']
12 })
13 export class NgForComponent implements OnInit {
14
15   //Mảng object: Dùng *ngFor để duyệt mảng object
16   public DSNV:any = [{ten:"khai",tuoi:18},{ten:"long",tuoi:17},{ten:"minh",tuoi:22}]
17   //Mảng
18   public Colors = ["red","green","blue"]
19   constructor() { }
20
21   ngOnInit() {
22 }
```

*ngFor:

Ngoài ra để lấy được vị trí hay còn gọi là chỉ mục (**index**) của từng phần tử được duyệt ta có thể khai báo thêm biến để hứng lấy giá trị này.

```
@Component({
  selector: 'app-ngfor',
  template: `
    <div *ngFor="let cauhoi of DanhSachCauHoi; let STT = index">
      <h5><b>{{STT}} .</b> {{cauhoi}}</h5>
    </div>`,
  styleUrls: ['./ngfor.component.css']
})
export class NgforComponent implements OnInit {

  private DanhSachCauHoi:Array<any> = ["Tên của bạn là gì ?",
                                           "Bạn bao nhiêu tuổi",
                                           "Bạn đến từ đâu ?"]

  constructor() {}
  ngOnInit() {
  }
}
```

Lấy vị trí phần tử của
mảng gán vào biến
chạy STT

Kết quả:

- **ngFor**

- 0 . Tên của bạn là gì ?
- 1 . Bạn bao nhiêu tuổi
- 2 . Bạn đến từ đâu ?

❖ Attribute – directives

*ngClass: Trong trường hợp áp dụng 1 hoặc 2 class trên một div là **classA** và **classB** thì ta sử dụng như sau.

```
2 @Component({
3   selector: 'app-attribute',
4   template: `<div [ngClass] = "{mauchu:mauchu,fontchu:fontchu}"> cybersoft </div>`,
5   styles: [
6     `
7       .mauchu{
8         color:red;
9       }
10      .fontchu{
11        font-size:25px;
12      }
13    ]
14  })
15)
16 export class AtributeComponent implements OnInit {
17   public mauchu:boolean = true;
18   public fontchu:boolean = true;
19   constructor() { }
20   ngOnInit() {
21 }
```

*ngClass: Điều kiện của class nào = true thì class đó được thêm vào

❖ Bài tập:

Yêu cầu:

- + Tạo 1 component chứa form đăng nhập và đăng ký như hình.
- + bấm vào nút đăng nhập hiển thị form đăng nhập, bấm nút đăng ký hiển thị form đăng ký
- + Xây dựng phương thức click nếu username = cybersoft và password = cybersoft thì đăng nhập thành công thay đổi trạng thái status. Đồng thời ẩn form đi và hiện ra text xin chào.
(Gợi ý: Dựa vào biến status và *ngIf để thực hiện ẩn hiện).

The image displays two screenshots of a user interface component for login and registration. Both screenshots feature a dark background with white text fields and green buttons.

Screenshot 1 (Top): Shows a top navigation bar with "Đăng Nhập" (Login) and "Đăng Ký" (Register) buttons. Below the navigation are two input fields labeled "Tài Khoản:" and "Mật Khẩu:". A large green button labeled "Đăng nhập" (Login) is positioned below the password field.

Screenshot 2 (Bottom): Shows a similar layout but with different fields. It includes "Tài Khoản:", "Mật Khẩu:", "Email:", and a dropdown menu labeled "Công Việc" (Work). The dropdown is currently set to "Sinh Viên" (Student). A green "Đăng Ký" (Register) button is located at the bottom right.

❖ Attribute - Directive

*Attribute directive (Tự định nghĩa): Để định nghĩa 1 attribute directive cũng giống như định nghĩa 1 component tuy nhiên directive thì không có thành phần template hay css.

```
ts high-light-directive.directive.ts x
1  import { Directive, ElementRef, OnInit } from '@angular/core';
2
3  @Directive({
4      selector: '[appHighLightDirective]' Lớp đối tượng dùng để DOM
5  })
6  export class HighLightDirective implements OnInit{
7
8      constructor(private elementRef: ElementRef) {
9          Khai báo đối tượng đại diện cho thành phần chứa directive
10     }
11     ngOnInit(){
12         this.elementRef.nativeElement.style.backgroundColor = 'green';
13     }
14 }
```

Cú pháp dùng để DOM tương tự JS : Gán background = màu xanh lục

Cách Sử dụng: Muốn sử dụng directive ở module nào thì import và declaration lại module đó, từ đó có thể sử dụng ở bất kì component nào trong module đó.

```
<div appHighLightDirective> Cybersoft </div>
```

❖Attribute – Directive (DOM)

***Renderer2**: 1 tính năng của angular 4 dùng để DOM đến các thành phần của HTML thông qua các phương thức rõ ràng và cụ thể hơn.

```
import { Directive, ElementRef ,OnInit , Renderer2 } from '@angular/core';

@Directive({
  selector: '[appHighLightDirective]'
})
export class HighLightDirectiveDirective implements OnInit{

  constructor(private elementRef:ElementRef, private render:Renderer2) {

  }
  ngOnInit(){
    this.elementRef.nativeElement.style.backgroundColor = 'green';
    //Đối tượng Renderer2 giúp dom đến element nhanh chóng và cụ thể (dễ dàng) hơn
    //Tương tự code phía trên
    this.render.setStyle(this.elementRef.nativeElement, 'background-color','green');
  }
}
```

❖Attribute – Directive (Event)

*HostListener ('Tên sự kiện'): Dùng để định nghĩa sự kiện cho directive attribute.

```
export class HighLightDirective implements OnInit{  
  constructor(private elementRef: ElementRef, private render: Renderer2) {  
    } Định nghĩa sự kiện tương tự JQUERY  
    @HostListener('mouseenter') SuKienHover(eventData:Event)  
    {  
      this.render.setStyle(this.elementRef.nativeElement, 'background-color','green');  
    }  
    @HostListener('mouseleave') SuKienMouseLeave(eventData:Event)  
    {  
      this.render.setStyle(this.elementRef.nativeElement, 'background-color','blue');  
    }  
}
```

❖Attribute – Directive (Event)

***HostBinding** ('thuộc tính của đối tượng html'): Dùng để ràng buộc thuộc tính của đối tượng html chứa attribute directive.

```
//Ràng buộc thuộc tính background Color của Element
@HostBinding('style.backgroundColor') bgColor:string = 'red';

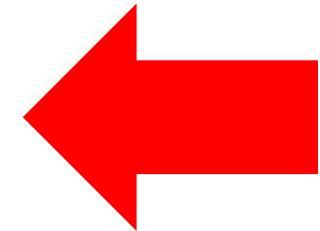
@HostListener('mouseenter') SuKienHover(eventData:Event)
{
    this.bgColor = 'green' ;
}

@HostListener('mouseleave') SuKienMouseLeave(eventData:Event)
{
    this.bgColor = 'red';
}
```

❖ Multiple Structural Directives

Khi ta sử dụng nhiều Structural directives trên cùng 1 tag như hình bên dưới thì sẽ bị lỗi.

```
<div class="lesson" *ngIf="lessons"
      *ngFor="let lesson of lessons">
    <div class="lesson-detail">
      {{lesson | json}}
    </div>
</div>
```



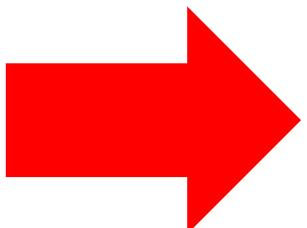
Lỗi

❖ Multiple Structural Directives

Do vậy ta phải tách ra dùng thêm 1 div ở ngoài để kiểm tra điều kiện. Nhưng nếu làm như vậy thì trên giao diện ta sẽ dư 1 thẻ div.

```
<div *ngIf="lessons">  
  <div class="lesson" *ngFor="let lesson of lessons">  
    <div class="lesson-detail">  
      {{lesson }}  
    </div>  
  </div>  
</div>
```

ng-container giúp ta sử dụng kết hợp được các structural directive nhưng không cần phải tồn thêm 1 tag hiển thị trên giao diện.



```
<ng-container *ngIf="lessons">  
  <div class="lesson" *ngFor="let lesson of lessons">  
    <div class="lesson-detail">  
      {{lesson }}  
    </div>  
  </div>  
</ng-container>
```

❖ Ng-container & *ngTemplateOutlet

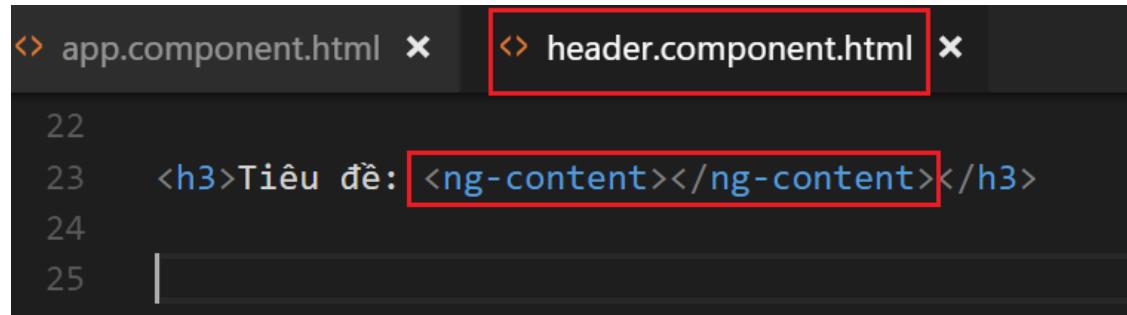
```
<ng-container *ngTemplateOutlet="template"> </ng-container>
<ng-template #template>
    <div> Nội dung template </div>
</ng-template>
```

Để sử dụng template bất kỳ đâu trên giao diện ta dùng **ng-container** + ***ngTemplateOutlet**

❖ ng-content

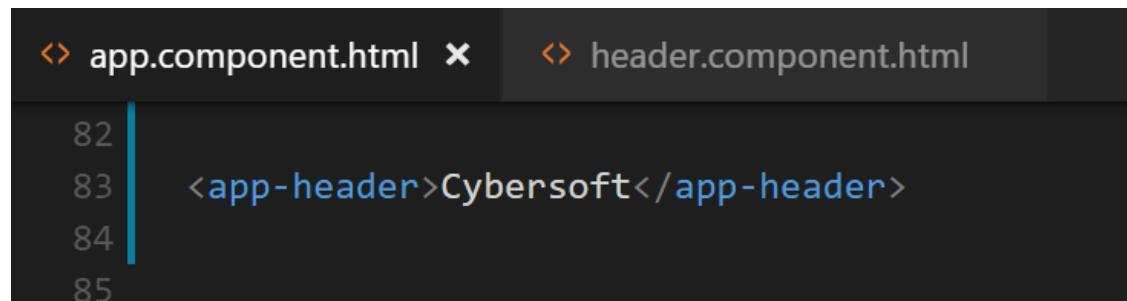
Sử dụng khi ta muốn chèn nội dung từ component cha đến component con tại vị trí chỉ định.

Tại header component



```
22
23 <h3>Tiêu đề: <ng-content></ng-content></h3>
24
25 |
```

Tại app component



```
82
83 <app-header>Cybersoft</app-header>
84
85 |
```

7. Input, Output, ViewChild

- **Input:** là thao tác truyền dữ liệu từ component cha vào component con (**component con là component khai báo input**).
- **Output:** Dùng để **đẩy giá trị** từ **component con ra ngoài component cha chứa nó** thông qua sự kiện (change, click, keyup, down,...) chuyển vào đối tượng EventEmiter rồi đưa ra ngoài.
- **Viewchild:** **Hiện thực hóa component bằng đối tượng trên class cha**, để sử dụng các **thuộc tính, phương thức** của **class component con**. Hay nói cách dễ hiểu là **Dom** đến thành phần con.

□ Input

- Là thao tác truyền dữ liệu từ component này (cha) sang component khác.

```
TS sinh-vien.component.ts ×  
1 import { Component, OnInit, Input } from '@angular/core';  
2  
3 @Component({  
4   selector: 'app-sinh-vien',  
5   template: `  
6     <div class="SinhVien">  
7       Họ tên: {{HoTen}} <br />  
8       Tuổi: {{Tuoi}}  
9     </div>  
10    `,  
11    styleUrls: ['./sinh-vien.component.css']  
12  })  
13 export class SinhVienComponent implements OnInit {  
14   @Input() HoTen:string;  
15   @Input() Tuoi:string;  
16   constructor() { }  
17  
18   ngOnInit() {  
19   }  
20 }  
21 }
```

```
TS sinh-vien.component.ts × TS danh-sach-sinh-vien.component.ts ×  
1 import { Component, OnInit } from '@angular/core';  
2  
3 @Component({  
4   selector: 'app-danh-sach-sinh-vien',  
5   template: `<app-sinh-vien *ngFor="let sv of DanhSachSinhVien"  
6     [HoTen]="sv.Ho_Ten"  
7     [Tuoi]="sv.Tuoi"></app-sinh-vien>`,  
8   styleUrls: ['./danh-sach-sinh-vien.component.css']  
9 })  
10 export class DanhSachSinhVienComponent implements OnInit {  
11  
12   private DanhSachSinhVien = [  
13     {Ho_Ten: "Nguyễn văn a", Tuoi:15},  
14     {Ho_Ten: "Nguyễn văn b", Tuoi:18},  
15     {Ho_Ten: "Nguyễn văn c", Tuoi:19}  
16   ]  
17   constructor() { }  
18  
19   ngOnInit() {  
20   }  
21 }  
22 }
```

Dữ liệu được truyền từ component a => b thông qua tham số **name** và **age**

Output

Component con (GheNgoicomponent)

```
@Component({
  selector: 'ghe-ngoi',
  template: `
    <button class="ghe chuatd" [ngClass]="{chuadat:!trangthai,dadat:trangthai}" (click)="DatGhe(trangthai)"></button>
    ,
    styles: [
      .ghe{
        width:50px;
        height:50px;
      }
      .dadat
      {
        background-color:red;
      }
      .chuadat
      {
        background-color:green;
      }
    ]
  )
})
```

Component xác định class đã đặt hay chưa đặt dựa vào giá trị biến của biến trangthai

```
export class GhengoiComponent implements OnInit {

  public trangthai = false;

  @Output () eventDatGhe = new EventEmitter();

  DatGhe(value:boolean)  Để đẩy được giá trị biến (khi thay đổi)
  {
    if(value == true)   ra ngoài ta phải thông qua 1 sự kiện
    {
      this.trangthai =false;
    }else
    {
      this.trangthai =true;
    }
    //Đây biến trạng thái ra component cha khi có sự thay đổi
    this.eventDatGhe.emit(this.trangthai);
  }

  public setname(name:string)
  {
    console.log(name);
  }
  {
    background-color:orange;
  }
}
})
```

Giá trị biến đẩy ra được truyền vào phương thức **emit()**

- Ta định nghĩa 1 component ghép ngồi.
- Trong component ghép ngồi ta định nghĩa trạng thái của ghế:
 - + Nếu trạng thái ghế là true: ghế chưa đặt (áp dụng css class chưa đặt)
 - + Nếu trạng thái ghế là false: ghế đã được đặt (áp dụng css class ghế đã đặt)
- Thông qua sự kiện click ta thay đổi giá trị trạng thái của ghế. Và ta muốn để giá trị này ra ngoài component danh-sach-ghe. Để làm gì để component đó tính được bao nhiêu ghế đã đặt và bao nhiêu ghế chưa đặt

Component cha (DanhSachGheNgoiComponent)

```
import { Component, OnInit, ViewChild, ViewChildren, QueryList } from '@angular/core';
import { GhengoiComponent } from '../ghengoi/ghengoi.component';

@Component({
  selector: 'danh-sach-ghe-ngozi',
  template: `
    <ghe-ngozi #t *ngFor="let ghe of DanhSachGhe" (eventDatGhe) = 'Dat_Ghe_Parent($event,ghe.maghe)'></ghe-ngozi>
    <p>Tổng ghế chưa đặt: {{SoGheChuaDat}}</p>
    <p>Tổng số ghế đã đặt: {{SoGheDaDat}}</p>
  `,
  styleUrls: ['./c.component.css']
})
export class DanhsachghengoiComponent implements OnInit {

  public DanhSachGhe:Array<any> = [{maghe:"ghe1",trangthai:false},{maghe:"ghe2",trangthai:false},{maghe:"ghe3",trangthai:false}];
  public SoGheDaDat:number = 0;
  public SoGheChuaDat:number = this.DanhSachGhe.length;
  Dat_Ghe_Parent(trangthaighe:boolean,maghe) //Biến được gửi từ component con sang
  {
    alert(maghe);
    if(trangthaighe == true)
    {
      this.SoGheDaDat++;
      this.SoGheChuaDat--;
    }
    else
    {
      this.SoGheDaDat--;
      this.SoGheChuaDat++;
    }
  }
}
```

Để bắt được giá trị đây ra từ component con thông qua emitter khi này. Ta tạo 1 sự kiện trong thẻ component con <ghe-ngozi> (sự kiện này trùng tên biến với biến emitter @Output ta đã tạo). Ta tạo thêm 1 phương thức để hứng lấy giá trị trong phương thức Emit() khi này = cách truyền vào \$event.

1/ Ta có 3 object từ mảng DanhSachGhe
Cần 3 thể hiện của component con Ghengoi (3 thẻ <ghe-ngozi> </ghe-ngozi>)

3/ Tại hàm này ta dựa vào giá trị đẩy ra từ component con (<ghe-ngozi>) là giá trị của biến trangthai của ghế đó. để xử lý tăng giảm biến số lượng ghế đã đặt và số lượng ghế chưa đặt.



Tổng ghế chưa đặt: 2

Tổng số ghế đã đặt: 1

Đặc tả:

Tại sao ta lại dùng Output. Bởi vì khi ta tạo ra component ghế từ danh sách ghế ta đâu thê biết trước và định nghĩa bao nhiêu sự kiện click trên bao nhiêu cái ghế ? Một điều nữa 2 component này hiện tại code TypeScript đang độc lập với nhau thuộc 2 class đối tượng khác nhau. Do vậy không thể lấy được giá trị từ component kia khi nó click được. Nên cách duy nhất ta sử dụng là thông qua binding = Output().

□ View Child (DOM đến component)

Component con (DanhSachGheNgoiComponent)

```
export class DanhsachghengoiComponent implements OnInit {  
    public DanhSachGhe:Array<any> = [{maghe:"ghe1",trangthai:false},{maghe:"ghe2",trangthai:false},{maghe:"ghe3",trangthai:false}];  
    public SoGheDaDat:number = 0;  
    public SoGheChuaDat:number = this.DanhSachGhe.length;  
    Dat_Ghe_Parent(trangthaighe:boolean,maghe) //Biến được gửi từ component con sang  
    {  
        alert(maghe);  
        if(trangthaighe == true)  
        {  
            this.SoGheDaDat++;  
            this.SoGheChuaDat--;  
        }  
        else  
        {  
            this.SoGheDaDat--;  
            this.SoGheChuaDat++;  
        }  
    }  
    themdsuhe(maghe:string,trangthai:boolean)  
    {  
        var ob = {maghe:maghe,trangthai:trangthai};  
        this.DanhSachGhe.push(ob);  
    }  
}
```

Viewchild thêm ghế



change

Tổng ghế chưa đặt: 3

Tổng số ghế đã đặt: 0

Mã ghế: Thêm ghế



Tạo thêm 1 textbox và 1 button
thêm ghế
+ Hàm xử lý sự kiện

□ View Child (DOM đến component)

Component cha (QuanLyDanhSachGheNgoiComponent)

```
ts interactivecomponent.module.ts      ts quanlydanh sachghengoi.component.ts x  ts danh sachghengoi.com  
1 import { Component, OnInit, ViewChild } from '@angular/core';  
2 import { DanhsachghengoiComponent } from './danhsachghengoi/danhsachghengoi.component';  
3 @Component({  
4   selector: 'quanly-danhsachghe',  
5   template: `  
6     <h3>Viewchild thêm ghế</h3>  
7     <danh-sach-ghe-ngozi></danh-sach-ghe-ngozi>  
8     Mã ghế: <input #maghe>  
9     <button (click)="themghe(maghe.value)">Thêm ghế</button>  
10    `,  
11    styleUrls: ['./quanlydanh sachghengoi.component.css']  
12  })  
13  export class Quanlydanh sachghengoiComponent implements OnInit {  
14  
15    constructor() {}  
16  
17    @ViewChild (DanhsachghengoiComponent) dsgheCom:DanhsachghengoiComponent;  
18    themghe(maghe:string)  
19    {  
20      console.log(this.dsgheCom);  
21      this.dsgheCom.themdsghe(maghe,true);  
22    }  
23    ngOnInit() {  
24    }  
25  }  
26 }  
27
```

Tại đây thông qua **viewchild** để điều hướng đến **component con** gọi phương thức **themghe**(phương thức của component con)

component con

Kiểu dữ liệu component con

Giao diện của component con khi được component cha gọi

Viewchild thêm ghế



change

Tổng ghế chưa đặt: 3

Tổng số ghế đã đặt: 0

Mã ghế: **Thêm ghế**



Phương thức thêm ghế tại component cha làm nhiệm vụ gọi đến phương thức them ghế vào component con

☐ View Child (DOM đến tag html)

Ví Dụ 1: Dùng ViewChild Dom đến 1 thẻ thông thường có đánh #name

Import vào ViewChild và ElementRef từ angular core

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

Đặt #name cho thẻ cần DOM

```
<input type="text" (change)='ChangeName()' value="FullName" #inputFullName /> |
```

Khai báo kiểu dữ liệu cho thuộc tính DOM từ #name là ElementRef

```
@ViewChild('inputFullName') inputFullName: ElementRef  
ChangeName()  
{  
  console.log(this.inputFullName.nativeElement);  
}
```

Để lấy được thuộc tính và phương thức giống như javascript ta chấm đến thuộc tính nativeElement.

Ví dụ để lấy value:

this.inputFullName.nativeElement.value

Bài tập

ĐẶT VÉ XE BUS HÃNG CYBERSOFT

Tài xế

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32
33	34	35	36

Danh sách ghế đã đặt (5)

Ghế : số 7 \$100 [Hủy]

Ghế : số 4 \$100 [Hủy]

Ghế : số 12 \$100 [Hủy]

Ghế : số 16 \$100 [Hủy]

Ghế : số 20 \$100 [Hủy]



```
{SoGhe:1,TenGhe: "số 1", Gia:100, TrangThai:false},  
{SoGhe:2,TenGhe: "số 2", Gia:100, TrangThai:false},  
{SoGhe:3,TenGhe: "số 3", Gia:100, TrangThai:false},  
{SoGhe:4,TenGhe: "số 4", Gia:100, TrangThai:false},  
{SoGhe:5,TenGhe: "số 5", Gia:100, TrangThai:false},  
{SoGhe:6,TenGhe: "số 6", Gia:100, TrangThai:false},  
{SoGhe:7,TenGhe: "số 7", Gia:100, TrangThai:false},  
{SoGhe:8,TenGhe: "số 7", Gia:100, TrangThai:false},  
{SoGhe:9,TenGhe: "số 9", Gia:100, TrangThai:false},  
{SoGhe:10,TenGhe: "số 10", Gia:100, TrangThai:false},  
{SoGhe:11,TenGhe: "số 11", Gia:100, TrangThai:false},  
{SoGhe:12,TenGhe: "số 12", Gia:100, TrangThai:false},  
{SoGhe:13,TenGhe: "số 13", Gia:100, TrangThai:false},  
{SoGhe:14,TenGhe: "số 14", Gia:100, TrangThai:false},  
{SoGhe:15,TenGhe: "số 15", Gia:100, TrangThai:false},  
{SoGhe:16,TenGhe: "số 16", Gia:100, TrangThai:false},  
{SoGhe:17,TenGhe: "số 17", Gia:100, TrangThai:false},  
{SoGhe:18,TenGhe: "số 18", Gia:100, TrangThai:false},  
{SoGhe:19,TenGhe: "số 19", Gia:100, TrangThai:false},  
{SoGhe:20,TenGhe: "số 20", Gia:100, TrangThai:false},  
{SoGhe:21,TenGhe: "số 21", Gia:100, TrangThai:false},  
{SoGhe:22,TenGhe: "số 22", Gia:100, TrangThai:false},  
{SoGhe:23,TenGhe: "số 23", Gia:100, TrangThai:false},  
{SoGhe:24,TenGhe: "số 24", Gia:100, TrangThai:false},  
{SoGhe:25,TenGhe: "số 25", Gia:100, TrangThai:false},  
{SoGhe:26,TenGhe: "số 26", Gia:100, TrangThai:false},  
{SoGhe:27,TenGhe: "số 27", Gia:100, TrangThai:false},  
{SoGhe:28,TenGhe: "số 28", Gia:100, TrangThai:false},  
{SoGhe:29,TenGhe: "số 29", Gia:100, TrangThai:false},  
{SoGhe:30,TenGhe: "số 30", Gia:100, TrangThai:true},  
{SoGhe:31,TenGhe: "số 31", Gia:100, TrangThai:false},  
{SoGhe:32,TenGhe: "số 32", Gia:100, TrangThai:false},  
{SoGhe:33,TenGhe: "số 33", Gia:100, TrangThai:false},  
{SoGhe:34,TenGhe: "số 34", Gia:100, TrangThai:false},  
{SoGhe:35,TenGhe: "số 35", Gia:100, TrangThai:false},
```

Form – Form Validation (Form)

Dùng để hỗ trợ việc lấy giá trị từ **Form** người dùng **nhập vào** đưa vào biến **model** (thuộc tính kiểu dữ liệu object ta tạo trong class của component đó). Trong phần two-way binding chúng ta đã thao tác với **FormsModule** 1 lần rồi.

Form - validation

The screenshot shows a form with three input fields and one button. The first field is labeled 'Email' and contains the placeholder 'Email'. The second field is labeled 'Name' and contains the placeholder 'Name'. The third field is labeled 'School' and contains the placeholder 'School'. Below these fields is a 'Submit' button.

```
37 export class FormvalidationComponent implements OnInit {  
38     public schools:any = [{id:'1',name:'cybersoft'}, {id:'1',name:'myclass'}];  
39  
40     constructor() { }  
41     Submit(value:any)  
42     {  
43         console.log(value);  
44     }  
45     ngOnInit() {  
46     }  
47 }  
48 }
```

```
<form #registerForm="ngForm" class="form-horizontal" (ngSubmit) = "Submit(registerForm.value)">  
    <div class="form-group">  
        <label for="inputEmail3" class="col-sm-2 control-label">Email</label>  
        <div class="col-sm-10">  
            <input type="email" class="form-control" id="inputEmail3" name="email" placeholder="Email" ngModel>  
        </div>  
    </div>  
    <div class="form-group">  
        <label for="Name" class="col-sm-2 control-label">Name</label>  
        <div class="col-sm-10">  
            <input type="text" class="form-control" id="Name" name="name" placeholder="Name" ngModel>  
        </div>  
    </div>  
    <div class="form-group">  
        <label for="school" class="col-sm-2 control-label">School</label>  
        <div class="col-sm-10">  
            <select id="school" class="form-control" name="school" ngModel>  
                <option *ngFor='let school of schools' [value]=>{school.id}>{{school.name}}</option>  
            </select>  
        </div>  
    </div>  
    <div class="form-group">  
        <div class="col-sm-offset-2 col-sm-10">  
            <button type="submit" class="btn btn-default">Submit</button>  
        </div>  
    </div>  
</form>
```

1/ Đối tượng **ngForm** sẽ lấy các giá trị từ các **ngModel** thông qua thuộc tính **name**.
2/ Sau đó ta sẽ dùng sự kiện **(submit)** gọi đến phương thức xử lý do chúng ta định nghĩa: gửi vào tham số là đối tượng **ngForm** (biến **registerForm.value**)

□ Form – Form Validation

- Trong angular 6 hỗ trợ ta 1 module dành cho việc nhập liệu Form và kiểm tra giá trị đó là: **FormsModule**.

```
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormvalidationComponent } from './formvalidation/formvalidation.component';
4 //Import FormsModule
5 import {FormsModule} from '@angular/forms';import FormsModule tại module cần sử dụng FormValidation
6 @NgModule({
7   imports: [
8     CommonModule, FormsModule
9   ],
10  exports:[FormvalidationComponent],
11  declarations: [FormvalidationComponent]
12 })
13 export class FormvalidationModule { }
```

□ Form – Validation (Validation)

❖ Trạng thái của giá trị đầu vào thẻ input

- : **untouched** : Trường này chưa được chạm vào.
- : **touched** : Trường này đã được chạm vào.
- : **pristine** : Trường này chưa được thay đổi.
- : **dirty** : Trường này đã được thay đổi.
- : **invalid** : Trường có nội dung không hợp lệ.
- : **valid** : Trường có nội dung hợp lệ.

Giá trị trả lại khi ta kiểm tra các trạng thái sẽ là **true**. Nếu ngược lại thì sẽ là **false**

□ Form – Validation (Validation)

❖ Kiểm tra rỗng với required

Lưu ý: Ví dụ trên chụp còn thiếu ngModel trong the input

```
<input type="email" class="form-control" id="email" name="email" placeholder="Email" #email='ngModel' required  
  !-- Kiểm tra rỗng -->  
<div *ngIf="email.errors && (email.dirty || email.touched)">  
  <div *ngIf="email.errors.required">  
    Email không được rỗng !  
  </div>  
</div>
```

Nếu có sự **thay đổi** của thẻ input#email hoặc #email được chạm vào. && #email errors (required, không đúng định dạng, quá ký tự ...) thì sẽ hiển thị div này.
Ví dụ (true && true = true) nên hiển thị

Xét trong lỗi đó có required không nếu có thì hiển thị div báo lỗi này.

□ Form – Validation (Validation)

- ❖ Kiểm độ dài ký tự với **max-length & min-length**

```
<div class="col-sm-10">
  <input type="email" minlength='7' maxlength='15' #email='ngModel' required ngModel class="form-control"
    <! -- Kiểm tra rỗng -->
    <div *ngIf="email.errors && (email.dirty || email.touched)" class="alert alert-danger">
      <! -- Kiểm tra rỗng -->
      <div *ngIf="email.errors.required">
        Email không được rỗng !
      </div>
    <! -- Kiểm tra độ dài -->
    <div *ngIf = "(email.errors.minLength || email.errors.maxLength)">
      Độ dài từ 7 - 15 ký tự
    </div>
  </div>
</div>
```

Nếu độ dài nằm ngoài
khoảng
7<X<15 thì sẽ hiển thị
div này



□ Form – Validation (Validation)

❖ Kiểm tra định dạng chuỗi nhập vào với pattern

```
<input type="email" pattern="[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,3}$" minlength='7' maxlength='15' #email=''
<! -- Kiểm tra rỗng -->
<div *ngIf="email.errors && (email.dirty || email.touched)" class="alert alert-danger">
<! -- Kiểm tra rỗng -->
    <div *ngIf="email.errors.required">
        Email không được rỗng !
    </div>
<! -- Kiểm tra độ dài -->
    <div *ngIf = "(email.errors.minLength || email.errors.maxLength)">
        Độ dài từ 7 - 15 ký tự
    </div>
<! -- Kiểm tra định dạng chuỗi nhập liệu -->
    <div *ngIf = "email.errors.pattern">
        Email không đúng định dạng
    </div>
</div>
</div>
```



Kiểm tra định dạng với email thông qua thuộc tính pattern nếu đúng sẽ hiển thị div báo lỗi

Tham khảo thêm các pattern:

https://www.w3schools.com/tags/att_input_pattern.asp

❑ Gợi ý kiểm tra password trùng nhau

```
<div class="form-group">
<label class="control-label col-sm-2" for="pass">pass:</label>
<div class="col-sm-10">
    <input type="password" class="form-control" id="pass" name="pass" #pass placeholder="Enter pass" ngModel>
</div>
</div>
<div class="form-group">
<label class="control-label col-sm-2" for="repass">re pass:</label>
<div class="col-sm-10">
    <input type="password" #repass [ngClass]="{'ng-invalid':(pass.value!=repass.value), 'ng-valid':!(pass.value!=repass.value)}">
    <div *ngIf = "repass.errors || pass.value!=repass.value" class="alert alert-danger">
        <div *ngIf = "pass.value!=repass.value">
            password không trùng|</div>
    </div>
</div>
</div>
```



Ta dùng 2 biến đại diện cho 2 input **#pass #repass**
+Kiểm tra 2 giá trị nó có khác (!=) nhau không nếu **khác nhau** thì
hiển thị div báo lỗi.
+Đồng thời tự xét tay thuộc tính **ng-invalid** hay **ng-valid** cho
input thông qua **property binding** thuộc tính directive **ngClass**.

□ Form – Validation (Validation)

❖ Trạng thái đầu vào của thẻ Form

`pristine` : Không có trường nào được thay đổi.

`dirty` : Có một trường hoặc nhiều trường đã được thay đổi.

`invalid` : Nội dung Form là không hợp lệ.

`valid` : Nội dung Form là hợp lệ.

- Tương tự như thẻ `input` thì các thuộc tính của Form cũng có giá trị trả lại là `true` hoặc `false`.

```
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-10">
    <button type="submit" *ngIf='!registerForm.submitted' [disabled]='!registerForm.valid' class="btn btn-defau
  </div>
</div>
```

 Form chưa submit hiển thị, submit
rồi thì ẩn đi

 Form chưa nhập hợp lệ thì không cho
click, nhập hợp lệ thì enabled lên

□ Form – Validation (Validation)

❖ Ngoài ra angular 2 còn cung cấp ta thêm 1 số class CSS để validate.

AngularJS tự động thêm các class CSS cho Form và các thẻ Input tùy thuộc vào trạng thái của chúng. Đồng thời nó sẽ tự xóa đi class đó khi dữ liệu hợp lệ.

➤ Đối với Input

`ng-untouched` : Trường này chưa được chạm vào.

`ng-touched` : Trường này đã được chạm vào.

`ng-pristine` : Trường này chưa được thay đổi.

`ng-dirty` : Trường này đã được thay đổi.

`ng-valid` : Trường có nội dung hợp lệ.

`ng-invalid` : Trường có nội dung không hợp lệ.

`ng-valid-key` : key tương ứng với mỗi validation. Ví dụ: `ng-valid-required`, trường đã có nội dung

`ng-invalid-key` : Ví dụ: `ng-invalid-required`, trường không có nội dung

➤ Đối với form

`ng-pristine` : Không có trường nào được thay đổi.

`ng-dirty` : Có một trường hoặc nhiều trường đã được thay đổi.

`ng-valid` : Nội dung Form là hợp lệ.

`ng-invalid` : Nội dung Form là không hợp lệ.

`ng-valid-key` : key tương ứng với mỗi validation. Ví dụ: `ng-valid-required`, có một hoặc nhiều hơn một trường đã có nội dung

`ng-invalid-key` : Ví dụ: `ng-invalid-required`, chưa trường nào có nội dung

➤ Ví dụ:

```
styles: [`  
  input.ng-invalid {  
    border-left:5px solid red;  
  }  
  input.ng-valid {  
    border-left:5px solid green;  
  }  
]
```

- ✓ Ta thêm vào khi input hợp lệ tất cả thì thẻ input có border left là màu xanh.
- ✓ Ngược lại nếu chưa hợp lệ thì thẻ input có border left là màu đỏ

➤ Kết quả: Form - validation

Email: Độ dài từ 7 - 15 ký tự
Email không đúng định dạng

Name:

Name:

➤ Cách set giá trị cho form từ Modal

Bước 1: Dùng viewchild dom tới Form đăng ký ngoài html thông qua local reference “ registerForm” đại diện cho thẻ form

Bước 2: Vì giá trị form trả về là 1 object, nên khi ta set giá trị cho form cũng phải đưa vào 1 object với key là name của các ô input, value là giá trị value của input

HTML

```
<form #registerForm="ngForm" class="form-horizontal" (ngSubmit) = "Submit(registerForm.value)">
  <div class="form-group">
    <label for="inputEmail3" class="col-sm-2 control-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="inputEmail3" name="email" placeholder="Email" ngModel>
    </div>
  </div>
  <div class="form-group">
    <label for="Name" class="col-sm-2 control-label">Name</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" id="Name" name="name" placeholder="Name" ngModel>
    </div>
  </div>
  <div class="form-group">
    <label for="school" class="col-sm-2 control-label">School</label>
    <div class="col-sm-10">
      <select id="school" class="form-control" name="school" ngModel>
        <option *ngFor='let school of schools' [value]="school.id">{{school.name}} </option>
      </select>
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="submit" class="btn btn-default">Submit</button>
    </div>
  </div>
</form>
```

1/ Đối tượng **ngForm** sẽ lấy các giá trị từ các **ngModel** thông qua thuộc tính name.
2/ Sau đó ta sẽ dùng sự kiện (**submit**) gọi đến phương thức xử lý do chúng ta định nghĩa: **registerForm.value**

TypeScript

```
}
```

```
export class DangkyComponent implements OnInit {
```

```
@ViewChild('registerForm') dangkyF:NgForm
```

```
SetValueUser(){
```

```
this.dangkyF.setValue({
  TaiKhoan:'hieu',
  MatKhau:'hieu',
  confirmPass:'hieu',
  HoTen:'hieu',
  Email:'hieu',
  SoDT:'hieu',
  MaLoaiNguoiDung:'KhachHang',
  gender:'famale'
})
```

```
}
```

Bài tập tổng hợp

Please Sign Up It's free and always will be.

The form consists of six input fields arranged in two rows. The first row contains 'First Name' and 'Last Name'. The second row contains 'Display Name', 'Email Address', 'Password', and 'Confirm Password'. Below the form is a horizontal progress bar divided into five colored segments: green, orange, red, purple, and blue. A large blue button labeled 'Register' is positioned at the bottom left of the form area.

First Name	Last Name
Display Name	Email Address
Password	Confirm Password

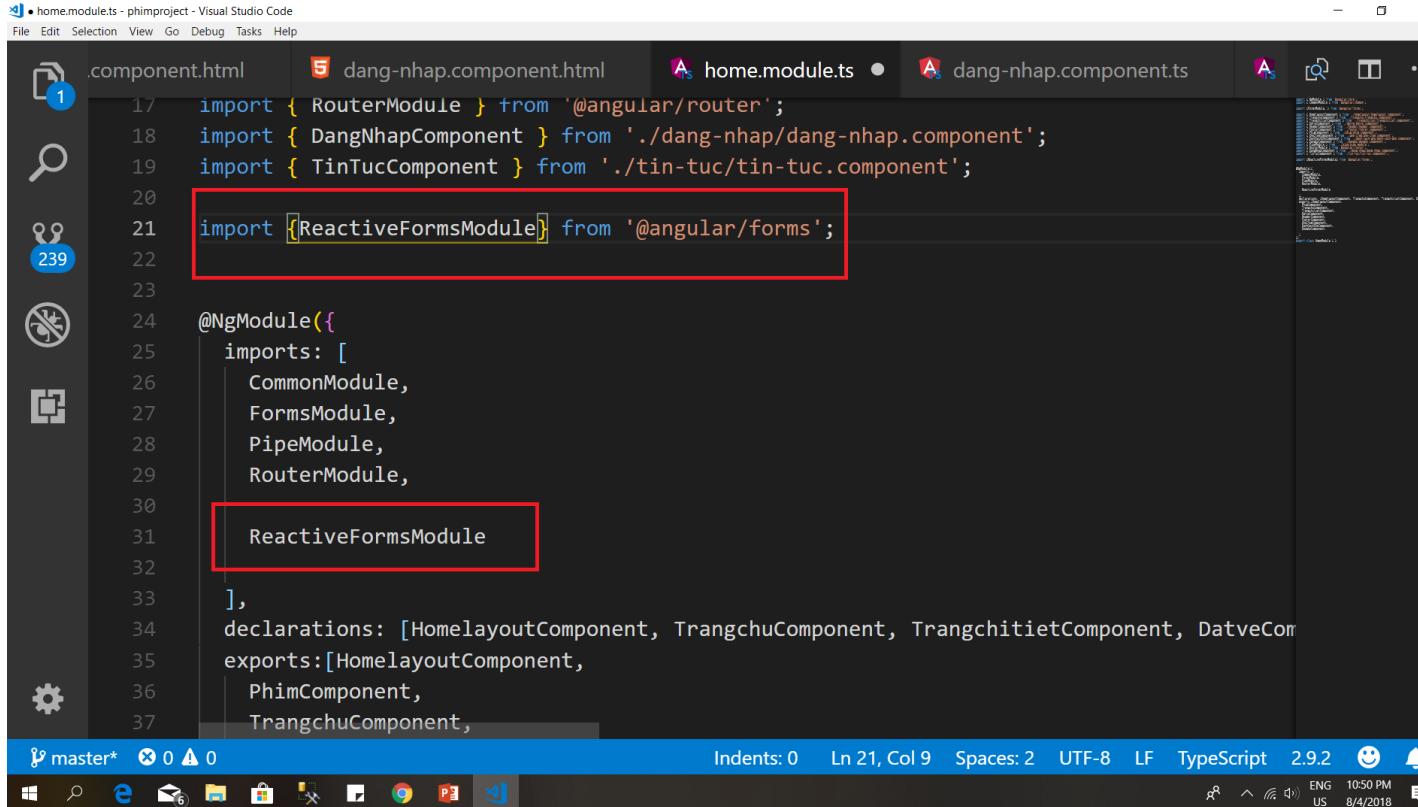
Register

☐ Tạo form như hình:

- ✓ Kiểm tra rỗng
- ✓ Kiểm tra tên từ 6 – 50 ký tự
- ✓ Kiểm tra định dạng email
- ✓ Kiểm tra password và confirm password
- ✓ Sau khi hợp lệ bấm register thì nó sẽ in ra table bên dưới. Với các cột tương ứng.
- ✓ Thêm 1 textbox để tìm kiếm theo họ tên hoặc email.
- ✓ Tạo thêm 1 component chứa Form đăng nhập với 2 input là email và password.
- ✓ Tạo 1 component hiển thị 2 link đăng ký, đăng nhập.
- ✓ Khi click vào link đăng ký thì form đăng ký hiển thị lên. Khi click vào link đăng nhập thì form đăng nhập hiển thị form đăng ký ẩn và ngược lại.
- ✓ Xây dựng chức năng cho form đăng nhập kiểm tra đăng nhập khi người dùng đăng nhập thành công xuất ra Xin chào email và ẩn form đó đi.

❑ ReactiveForm

- ❖ Ngoài FormsModule, Angular 6 hỗ trợ thêm các module để handle form, trong đó có ReactiveFormsModule.
- ❖ Cách sử dụng ReactiveFormsModule :
 - ❑ Bước 1: import ReactiveFormsModule vào module nào cần sử dụng



The screenshot shows the Visual Studio Code interface with the file `home.module.ts` open. The code imports several modules and defines an NgModule. A red box highlights the import statement for `ReactiveFormsModule` at line 21:

```
17  import { RouterModule } from '@angular/router';
18  import { DangNhapComponent } from './dang-nhap/dang-nhap.component';
19  import { TinTucComponent } from './tin-tuc/tin-tuc.component';
20
21  import { ReactiveFormsModule } from '@angular/forms';
22
23
24  @NgModule({
25    imports: [
26      CommonModule,
27      FormsModule,
28      PipeModule,
29      RouterModule,
30
31      ReactiveFormsModule
32
33    ],
34    declarations: [HomelayoutComponent, TrangchuComponent, TrangchitietComponent, DatveCor
35    exports:[HomelayoutComponent,
36      PhimComponent,
37      TrangchuComponent,
```

The status bar at the bottom indicates the file is on branch `master*`, has 0 changes, and is using TypeScript 2.9.2. The system tray shows various icons including a clock showing 10:50 PM on 8/4/2018.

□ ReactiveForm

- ❑ Bước 2: Xây dựng form Đăng Nhập ở giao diện HTML
 - ❑ Bước 3: Xây dựng ReactiveForm Đăng Nhập ở Modal (typescript)
 - ❑ Bước 4: Kết nối 2 form, dữ liệu người dùng nhập vào HTML được truyền vào ReactiveForm ở Modal
 - ❑ Bước 5: Viết phương thức DangNhap, console.log ra value của form

```
<div class="container">
  <div class="row">
    <div class="col-5 mx-auto">
      <form [formGroup]="formDangNhap" (ngSubmit)="DangNhap()">
        <h4 class="display-4">Đăng Nhập</h4>
        <div class="form-group">
          <label for="">Tài Khoản</label>
          <input type="text" class="form-control" formControlName="TaiKhoan" >
        </div>
        <div class="form-group">
          <label for="">Mật Khẩu</label>
          <input type="text" class="form-control" formControlName="MatKhau">
        </div>
        <div class="form-group text-center">
          <button type="submit" class="btn btn-success">Đăng nhập</button>
        </div>
      </form>
    </div>
  </div>
</div>
```

```
dang-nhap.component.ts - phim-project - Visual Studio Code
File Edit Selection View Go Debug Tasks Help

dang-nhap.component.html dangky.component.ts home.module.ts dang-nhap.component.ts

2 dang-nhap.component.html
5 import { AuthService } from '../../../../../services/auth.service';
6
7 import { FormGroup, FormControl } from '@angular/forms'; ← import từ angular forms
8
9 @Component({
10   selector: 'app-dang-nhap',
11   templateUrl: './dang-nhap.component.html',
12   styleUrls: ['./dang-nhap.component.css']
13 })
14 export class DangNhapComponent implements OnInit {
15
16   public formDangNhap:FormGroup; ← Khởi tạo ReactiveForm, Kiểu dữ liệu
17   là FormGroup
18
19   ngOnInit() {
20     this.formDangNhap = new FormGroup({
21       'TaiKhoan': new FormControl(null),
22       'MatKhau': new FormControl(null)
23     })
24   }
25   DangNhap(){
26     console.log(this.formDangNhap.value);
27   }
}

Ln 8, Col 1 Spaces: 2 UTF-8 LF TypeScript 2.9.2 ⚡ EN-US 11:27 PM 8/4/2018
```

☐ ReactiveForm

☐ Kết quả:

Đăng Nhập

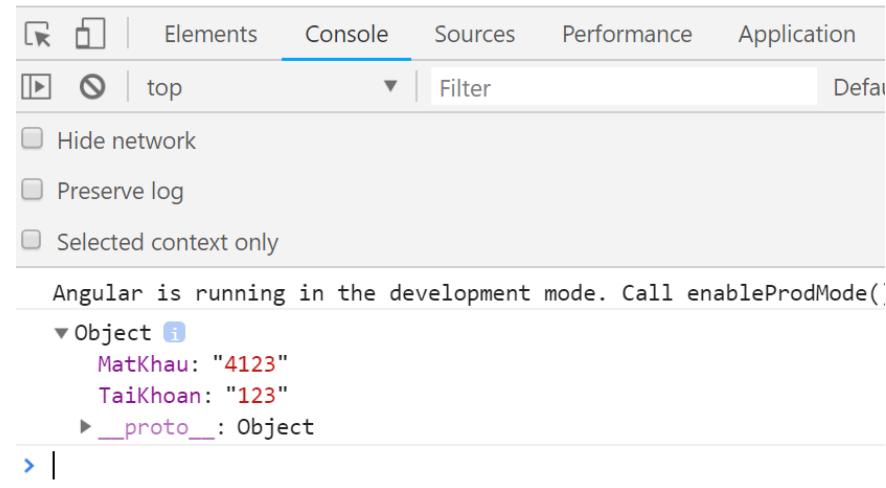
Tài Khoản

123

Mật Khẩu

4123

Đăng nhập



The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The console output displays the following message and an object:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.  
Object {  
  MatKhau: "4123"  
  TaiKhoan: "123"  
  __proto__: Object  
}
```

❑ Validation với ReactiveForm

- ❑ Ở HTML, kiểm tra không hợp lệ thì hiện alert

```
5  dang-nhap.component.html x  A dangky.component.ts  A home.module.ts  A dang-nhap.component.ts  A app.module.ts
6      <div class="display-4">Đang Nhập</div>
7      <div class="form-group">
8          <label for="">>Tài Khoản</label>
9          <input type="text" class="form-control" formControlName="TaiKhoan">
10
11         <div *ngIf="formDangNhap.get('TaiKhoan').invalid && formDangNhap.get('TaiKhoan').touched">
12             <div class="alert alert-danger" *ngIf="formDangNhap.get('TaiKhoan').errors['required']">
13                 | Vui lòng nhập tài khoản
14             </div>
15         </div>
16     </div>
17     <div class="form-group">
18         <label for="">Mật Khẩu</label>
19         <input type="text" class="form-control" formControlName="MatKhau">
20
21 
```

- ❑ Gắn validators trực tiếp vào formControl

```
public formDangNhap: FormGroup;
ngOnInit() {
    this.formDangNhap = new FormGroup({
        'TaiKhoan': new FormControl(null,Validators.required),
        'MatKhau': new FormControl(null,Validators.minLength(4))
    })
}

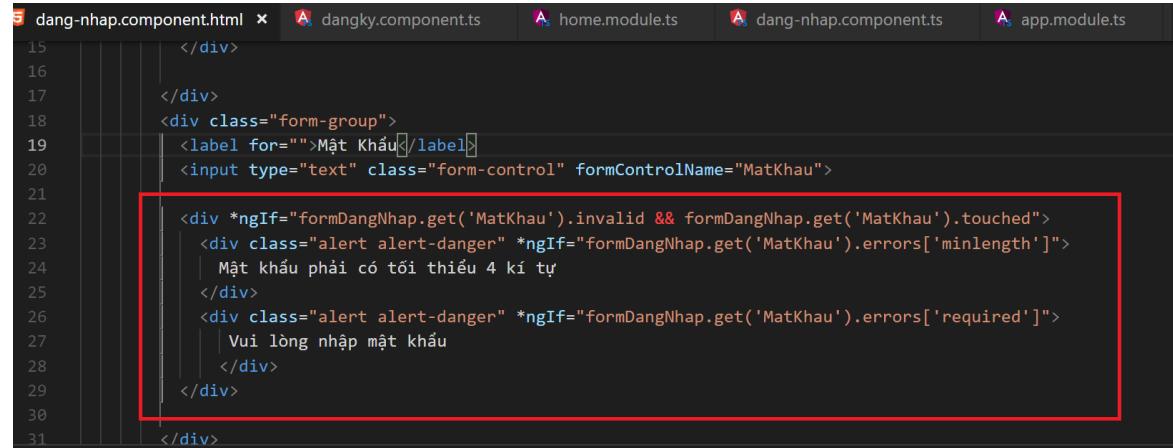
```

Gắn validation cho input tại đây,
Kiểm tra TaiKhoan không được
rỗng, MatKhau phải có độ dài
tối thiểu là 4 kí tự

```
15      </div>
16
17      </div>
18      <div class="form-group">
19          <label for="">Mật Khẩu</label>
20          <input type="text" class="form-control" formControlName="MatKhau">
21
22          <div *ngIf="formDangNhap.get('MatKhau').invalid && formDangNhap.get('MatKhau').touched">
23              <div class="alert alert-danger" *ngIf="formDangNhap.get('MatKhau').errors['minlength']">
24                  | Mật khẩu phải có tối thiểu 4 kí tự
25              </div>
26          </div>
27
28      </div>
29      <div class="form-group text-center">
30          <button type="submit" class="btn btn-success">Đăng nhập</button>
31      </div>
```

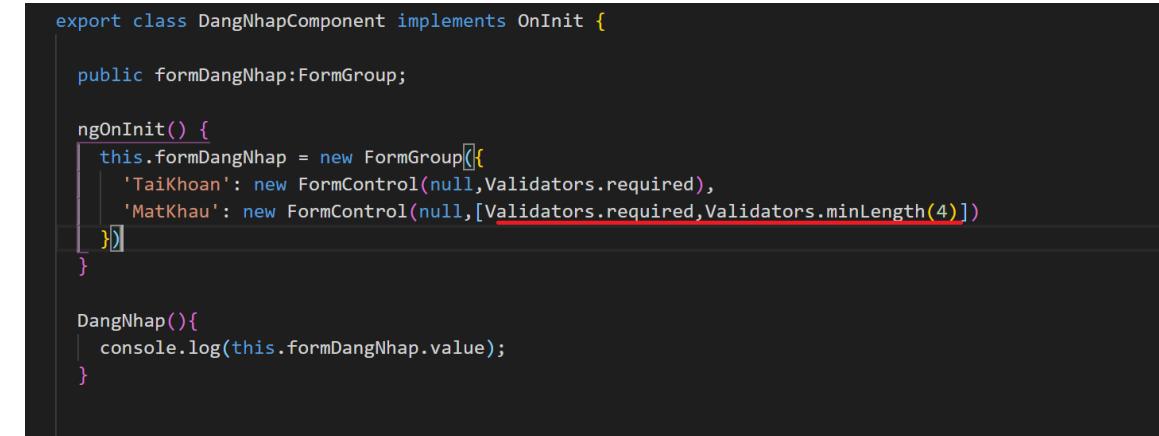
☐ Kết hợp nhiều validators

- ☐ Ở HTML, kiểm tra không hợp lệ thì hiện alert



```
15   </div>
16
17   </div>
18   <div class="form-group">
19     <label for="">Mật Khẩu</label>
20     <input type="text" class="form-control" formControlName="MatKhau">
21
22     <div *ngIf="formDangNhap.get('MatKhau').invalid && formDangNhap.get('MatKhau').touched">
23       <div class="alert alert-danger" *ngIf="formDangNhap.get('MatKhau').errors['minlength']">
24         | Mật khẩu phải có tối thiểu 4 kí tự
25       </div>
26       <div class="alert alert-danger" *ngIf="formDangNhap.get('MatKhau').errors['required']">
27         | Vui lòng nhập mật khẩu
28       </div>
29     </div>
30
31   </div>
```

- ☐ Gắn validators trực tiếp vào formControl



```
export class DangNhapComponent implements OnInit {
  public formDangNhap: FormGroup;
  ngOnInit() {
    this.formDangNhap = new FormGroup({
      'TaiKhoan': new FormControl(null,Validators.required),
      'MatKhau': new FormControl(null,[Validators.required,Validators.minLength(4)])
    })
  }
  DangNhap(){
    console.log(this.formDangNhap.value);
  }
}
```

Bài tập tổng hợp

#	No.	Hình ảnh	Tên Phim	Sản xuất	Ngày công chiếu	Trailer
<input type="checkbox"/>	0		jkhakjhajkshjdas	fgsdfgsdfg	2018-06-12T00:00:00	dsfgsdfgsdfg
<input type="checkbox"/>	1		Minions	Kyle Bald	2015-07-10T00:00:00	https://www.youtube.com/watch?v=Wfql_DoHRKc

ID	Title
Description	
Rating	Director
Release Date	Trailer
Image Url	
ADD MOVIE	

☐ Tạo form add phim như hình:

- ✓ Kiểm tra rỗng
- ✓ Kiểm tra description trên 20 kí tự
- ✓ Kiểm tra Title phải là chữ
- ✓ Kiểm tra Rating phải là số
- ✓ Sau khi hợp lệ bấm Add Movie , tiến hành thêm phim vào mảng DanhSachPhim
- ✓ Tạo bảng động hiển thị danh sách phim
- ✓ Tên phim hiển thị ra bảng nhỏ hơn 10 ký tự, lớn hơn 10 kí tự thì cắt bớt và thêm dấu “...”
- ✓ Ngày chiếu hiển thị theo định dạng Ngày- Tháng-Năm Giờ : phút : giây
- ✓ ID hiển hiện ra bảng phải là chữ in hoa.
- ✓ Thêm button Cập nhật, sửa lại phim.

Pipes

Là một filter của angular giúp format định dạng dữ liệu khi hiển thị ra màn hình

Loại pipe	Mô tả	Cú pháp
LowerCasePipe	Chuyển đổi từ hoa sang thường	{ {[gia trị] lowercase} }
UpperCasePipe	Chuyển đổi từ chữ thường => hoa	{ {[gia trị] uppercase} }
DatePipe	Dùng để định dạng ngày tháng năm	{ {today date:'fullDate'} }
PercentPipe	Định dạng phần trăm (Số đó * 100)	{ {value percent} }, { {value A.B-C} }
DecimalPipe	Định dạng tiền tệ, số liệu	{ {value percent} }, { {value A.B-C} }
jSonPipe	Chuyển từ 1 mảng json => chuỗi json	{ {objectJson json } }
jSonSlice	Lấy số phần tử từ X->Y-1 trong mảng	{ {objectJson slice:2:3} }

❖ Ví dụ:

```
ts pipe.component.ts x

1 import { Component, OnInit } from '@angular/core';
2 @Component({
3   selector: 'app-pipe',
4   template: `
5     <div> {{info | uppercase}} </div>
6     <div> {{info | lowercase}} </div>
7     <div> {{percent | percent}} </div>
8     <div> {{percent | percent:'2.3-5'}} </div>
9     <div> {{percent | number:'3.1-6' }} </div>
10    <div> {{objectJson | json }} </div>
11    <div> {{array | slice:1:5}} </div>
12  `,
13  styles: ['./pipe.component.css']
14})
15 export class PipeComponent implements OnInit {
16   private info:string = "Học viện đào tạo cybersoft";
17   private percent:number = 32321.962;
18   private objectJson:Object = {hoten:'Minh',lop:'frontend01',diem:[10,2,6]};
19   private array:Array<string> = ['pt1','pt2','pt3','pt4','pt5'];
20   constructor() { }
21   ngOnInit() {
22 }
```

<https://angular.io/docs/ts/latest/api/#?apiFilter=pipe&type=pipe> (rất nhiều pipe gồm định dạng format animation ...) tham khảo thêm.

❖ Thực hành: tạo pipe rút gọn chuỗi

B1: Trong folder **app**, tạo ra module **PipeModule** quản lý các pipe do ta tự định nghĩa

B2: Dùng cli tạo ra pipe với cú pháp: **ng g pipe shortcut**

B3: Định dạng nội dung của ShortcutPipe

```
import { PipeTransform, Pipe } from "@angular/core";

@Pipe({
  name: 'shortcut'
})
export class ShortcutPipe implements PipeTransform{
  transform(value, limit){
    return value.substr(0,limit)+"..."
  }
}
```

❖ Thực hành: tạo pipe rút gọn chuỗi

B4: Ở Pipe, tiến hành import và declaration và exports ShorcutePipe để PipeModule quản lý tương tự như 1 component (Angular Cli đã hỗ trợ)

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ShortcutPipe } from './shortcut.pipe';

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

B5: Import *PipeModule* vào module nào sử dụng *ShorcutePipe*

```
import { DangnhapComponent } from './dangnhap/dangnhap.component';
import { DangkyComponent } from './dangky/dangky.component';

import {PipeModule} from '../pipe/pipe.module';

@NgModule({
  imports: [
    CommonModule,RouterModule, PipeModule,FormsModule
  ],
  declarations: [TrangchuComponent, TrangchitietComponent, TrangdatgheComponent],
  exports: [TrangchuComponent, TrangchitietComponent, TrangdatgheComponent]
})
export class HomeModule { }
```

❖ Thực hành: tạo pipe rút gọn chuỗi

B6: Gắn ShortcutPipe vào thẻ HTML

```
<td>{{nguoIDung.MaNhom | shortcut:5 }}</td>
```

Với:

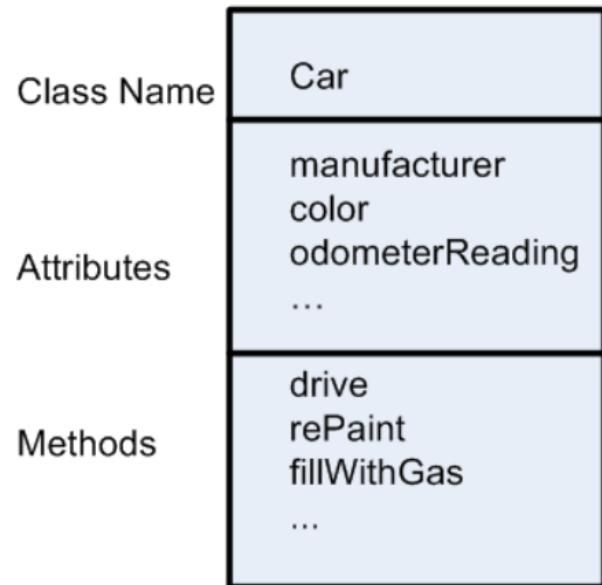
- shortcut là tên đại diện của pipe
- 5 là tham số limit
- nguoIDung.MaNhom là value đầu vào
- Giá trị return là cái sẽ được hiển thị lên giao diện

```
import { PipeTransform, Pipe } from "@angular/core";

@Pipe([
  name: 'shortcut'
])
export class ShortcutPipe implements PipeTransform{
  transform(value,limit){
    return value.substr(0,limit)+"..."
  }
}
```

Object Class

- ❖ Object Class (TypeScript đã học) là prototype



Cú pháp tạo Class

```
ng g class [tên class]
```

Observable

Observable là 1 tính năng mới của ES7. Trước khi nhắc đến observable ta cần tìm hiểu 1 tí về callback function và promise(ES6)

Ôn tập 1 tí về khái niệm bất đồng bộ trong javascript hay còn gọi là ajax (Asynchronous JavaScript and XML)

```
Function ShowMessage()
{
    var message = 'hello';
    $.ajax({
        url : "url",
        data : {}
        success : function(result){
            message = result;
        },
        error:function(error){
            message = error;
        }
    });
    alert(message);
}
```

Theo các bạn đoạn code trên sẽ trả về giá trị message là gì ?

- a. hello
- b. Giá trị result trong hàm success
- c. Giá trị error trong hàm error

Nói về callback function.

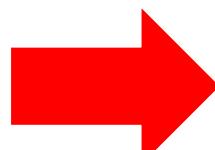
-Khái niệm: 1 function A được truyền vào 1 function B thông qua danh sách tham số của B. Lúc này tại hàm B sẽ gọi hàm A thực hiện một chức năng mà A đang nắm giữ. Đó là lý thuyết

Ví dụ: jQuery dùng rất nhiều callback function

```
$('selector').click(function(){
    alert('hello cybersoft');
    //function A đóng vai trò là function, function B là click ở đây là hàm click không có tham số
});
```

Cách viết khác hàm click nhận vào tham số là 1 function, function xử lý chức năng cho sự kiện click.

```
$('selector').click>ShowMessage);
function ShowMessage(){
    alert('hello cybersoft');
};
```

Đảm bảo mọi việc diễn ra theo đúng trình tự ví dụ khi click vào sự kiện 1 nút thì nút đó sẽ thực thi hành vi của 1 hàm khác. Jquery ứng dụng rất nhiều. Trong đó có ajax

Tương tự Promise (1 tính năng của ES6) cũng giống callback function

Promise được đưa vào Javascript từ ES6, đây có thể coi là một kỹ thuật nâng cao giúp xử lý vấn đề bất đồng bộ hiệu quả hơn. Trước đây kết quả của một tác vụ đồng bộ và bất đồng bộ sẽ trả về một kiểu dữ liệu nào đó hoặc thực hiện một **Callback Function**.

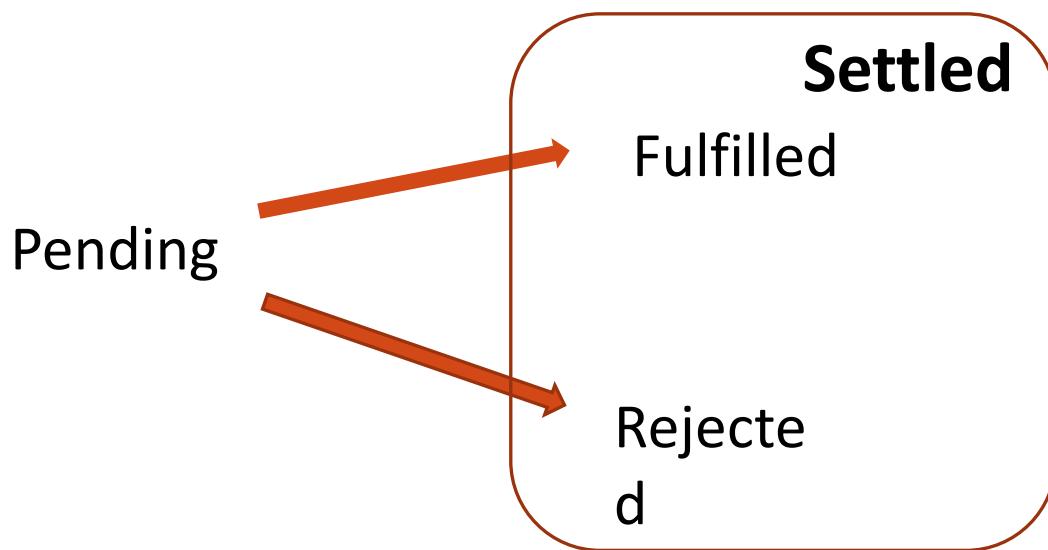
Với trường hợp thực hiện Callback Function thì sẽ dễ xảy ra lỗi Callback Hell (Trong callback function lại gọi callback function cứ như vậy các callback function sẽ lồng vào nhau rất khó quản lý khi có lỗi xảy ra trừ khi tự mình code).

Promise

Promise sinh ra để xử lý một kết quả của một hành động cụ thể, Kết quả của mỗi hành động sẽ có các trạng thái như

- + Thành Công làm gì (trạng thái Fulfilled)
- + Thất bại làm gì (trạng thái Rejected)
- + Chờ xử lý hoặc bị từ chối làm gì (Pending)

Nói cách khác nó cũng tựa như ajax khi này



Fulfilled Rejected gọi là **settled** tức là đã xử lý xong.

```

function ShowMessage()
{
    var promise = new promise(successData, failData);
    var message = 'hello';
    var request = new XMLHttpRequest();
    request.open('GET', 'url');
    request.onload = function() {
        if (request.status == 200) {
            //Hàm tham số đầu tiên trong biểu thức then (Thành công)
            successData(request.response,request.status);
        } else {
            //Hàm tham số đầu tiên trong biểu thức then (Thất bại)
            failData(request.statusText,request.status);
        }
    };
    //Gọi promise:
    promise.then(successData,failData).cache(function(){
        console.log("error")
    });
}

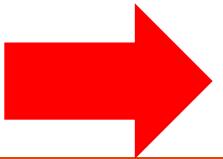
```

```

function successData(result,status)
{
    message = result; //(1)
    alert(message);
}

function failData(error,status)
{
    message = error; //(2)
    alert(error)
}

```

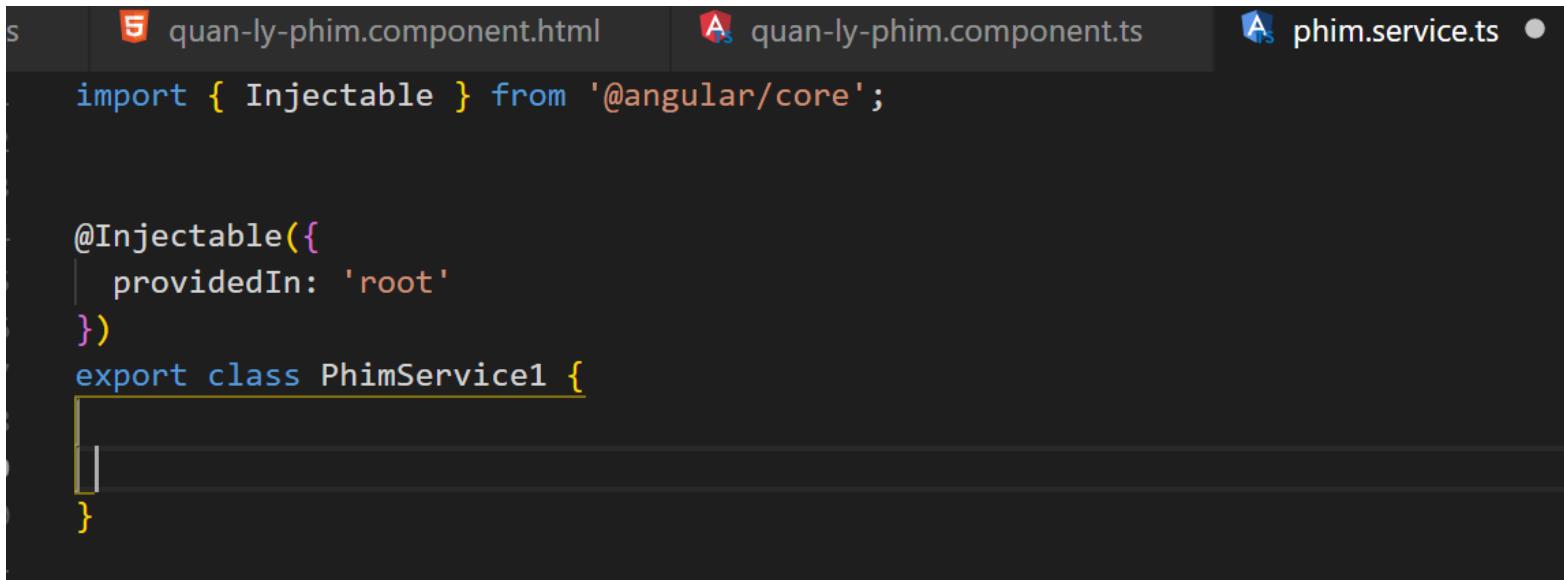
 Kết quả của 1 hàm promise sẽ trả về 1 đối tượng promise nên setup tất cả các hàm xử lý chỉ cần .then() để quản lý các promise.

Observable

- ❖ Observable cũng tương tự như promise nhưng **promise** nhận giá trị tại **1 thời điểm** là thành công nhảy vào hàm thành công làm trả ra kết quả hoặc thất bại cũng vậy.
- ❖ Observable nó lấy **nhiều giá trị** tại **nhiều thời điểm**.
- ❖ Ví dụ như bài toán đầu tiên message in ra là hello thay vì là 1 trong 2 giá trị tương lai nó sẽ trả về thì observable nó nhận cả 2 giá trị tại 2 thời điểm và binding lên model. Thêm 1 điều nữa request được gửi đi ở observable thì có thể hủy được.

□ Service - HTTPService

- Service là chứa các phương thức tương tác với backend(server) để lấy hoặc upload dữ liệu từ backend về thông qua các phương thức như GET, POST, PUT, DELETE và thư viện hỗ trợ từ angular 2 gọi là **Observable** để đổi vào đối tượng (Object).
- Nói nôm na service chứa các hàm gọi ajax để lấy dữ liệu về



The screenshot shows a code editor with three tabs: 'quan-ly-phim.component.html', 'quan-ly-phim.component.ts', and 'phim.service.ts'. The 'phim.service.ts' tab is active and displays the following code:

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

@Injectable({
  providedIn: 'root'
})
export class PhimService1 {
```

ng g service [tên service]

Sử dụng service

- Để sử dụng service cho component nào thì import service cho component đó.

- Ví dụ:

Tạo module TestServiceModule và component TestServiceComponent. Khi ta cần sử dụng service để lấy dữ liệu hiển thị thì:

- + Tại component sử dụng ta truyền vào tham số hàm khởi tạo service đó.

Service

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

@Injectable()
export class NhanvienService {
  //Danh sách nhân viên hiện tại là any sau này nó sẽ là kiểu dữ liệu của Class object
  private DanhSachNhanVien:Array<any> = [
    {manv:'01',hoten:'nhan vien 1',namsinh:'1993'},
    {manv:'02',hoten:'nhan vien 2',namsinh:'1991'},
    {manv:'03',hoten:'nhan vien 3',namsinh:'1992'},
    {manv:'04',hoten:'nhan vien 4',namsinh:'1990'}
  ]
  //Phương thức trong service
  public GetNhanVien():Array<any>
  {
    //Sau này tại đây ta sẽ liên kết với api từ backend (server) để lấy dữ liệu
    return this.DanhSachNhanVien;
  }
  constructor() { }
```

Component

```
import { Component, OnInit } from '@angular/core';
import { NhanvienService } from '../../service/nhanvien.service';

@Component({
  selector: 'app-testservice',
  template: `
    <div *ngFor='let nv of DanhSachNhanVien'>
      Mã NV:{{nv.manv}}, Họ tên: {{nv.hoten}}, Năm sinh: {{nv.namsinh}}
    </div>`,
  styleUrls: ['./testservice.component.css']
})
export class TestserviceComponent implements OnInit {
  public DanhSachNhanVien:Array<any>;
  constructor(nvService:NhanvienService) {
    //Gọi service để lấy dữ liệu danh sách nhân viên hiển thị ra component
    this.DanhSachNhanVien = nvService.GetNhanVien();
  }
}
```

HTTP (Http Module)

Http module cung cấp cho chúng ta service http dung để giao tiếp với backend(server) thông qua một số phương thức như get, post, put, delete ...

```
import {HttpModule} from '@angular/http';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, LayoutModule,DatabindingModule,Dir
  ,TestserviceModule,MovieModule,HttpModule
  ],
  providers: []
})
```

Để dùng được http service ta cần import **HttpModule** vào toàn ứng dụng thông qua **appmodule**

Get Post Put Delete(http service) – Lưu trữ cục bộ

Tương tự như jQuery angular 2 cũng hỗ trợ một số phương thức get post put ... Đã giới thiệu ở phần HTTP Module.

Đối với Backend các giao thức thường được sử dụng

Get: Thường dùng để lấy dữ liệu (Thực hành rồi)

Post: Thường dùng để tạo mới dữ liệu

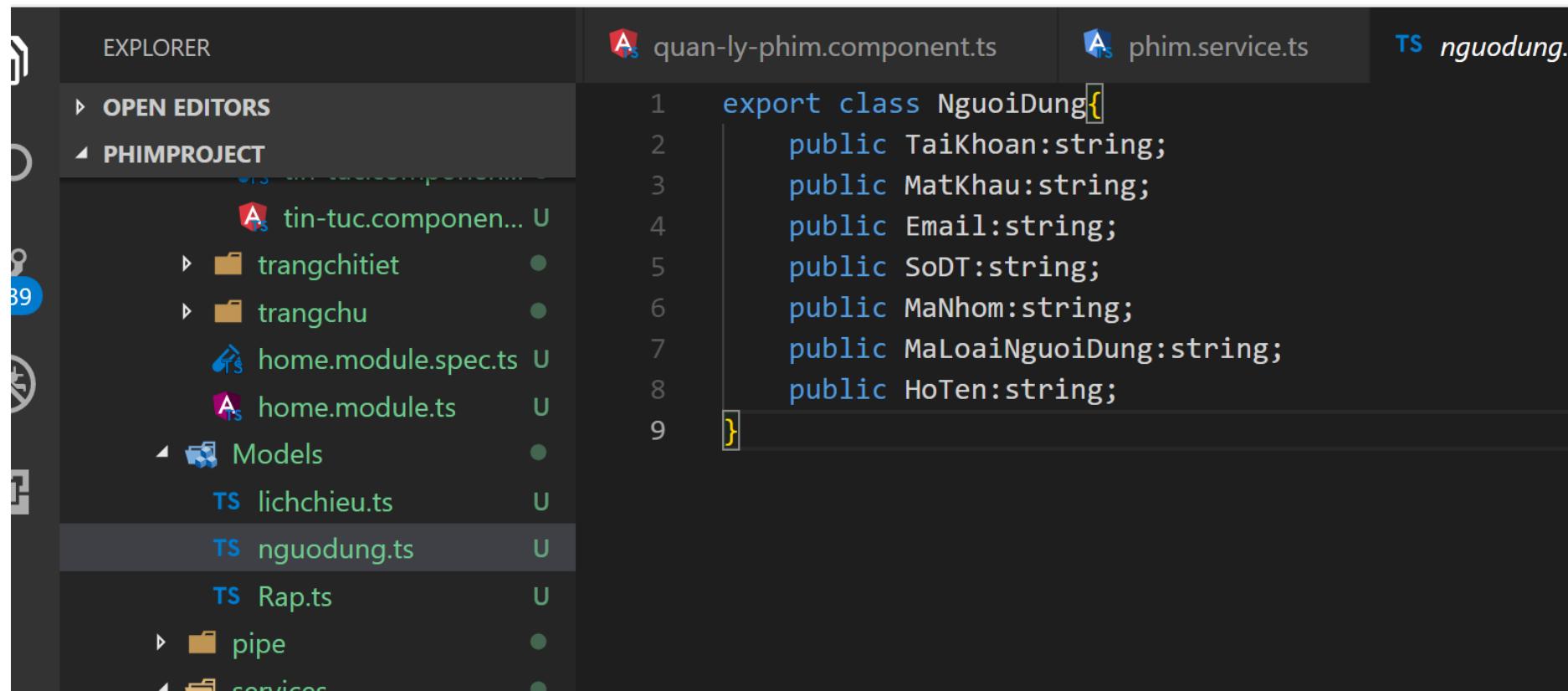
Put: Thường dùng để cập nhật dữ liệu

Delete: Thường dùng để xóa dữ liệu

❖ Thực hành: Chức năng đăng ký người dùng

B1: Trong folder **app**, tạo folder Models chứa các class lớp đối tượng.

B1: Tạo **class Object** chứa trong **Models** đặt tên **NguoiDung** (Để quản lý thông tin khách hàng bao gồm tài khoản , password ... thực hiện chức năng đăng ký, đăng nhập) .

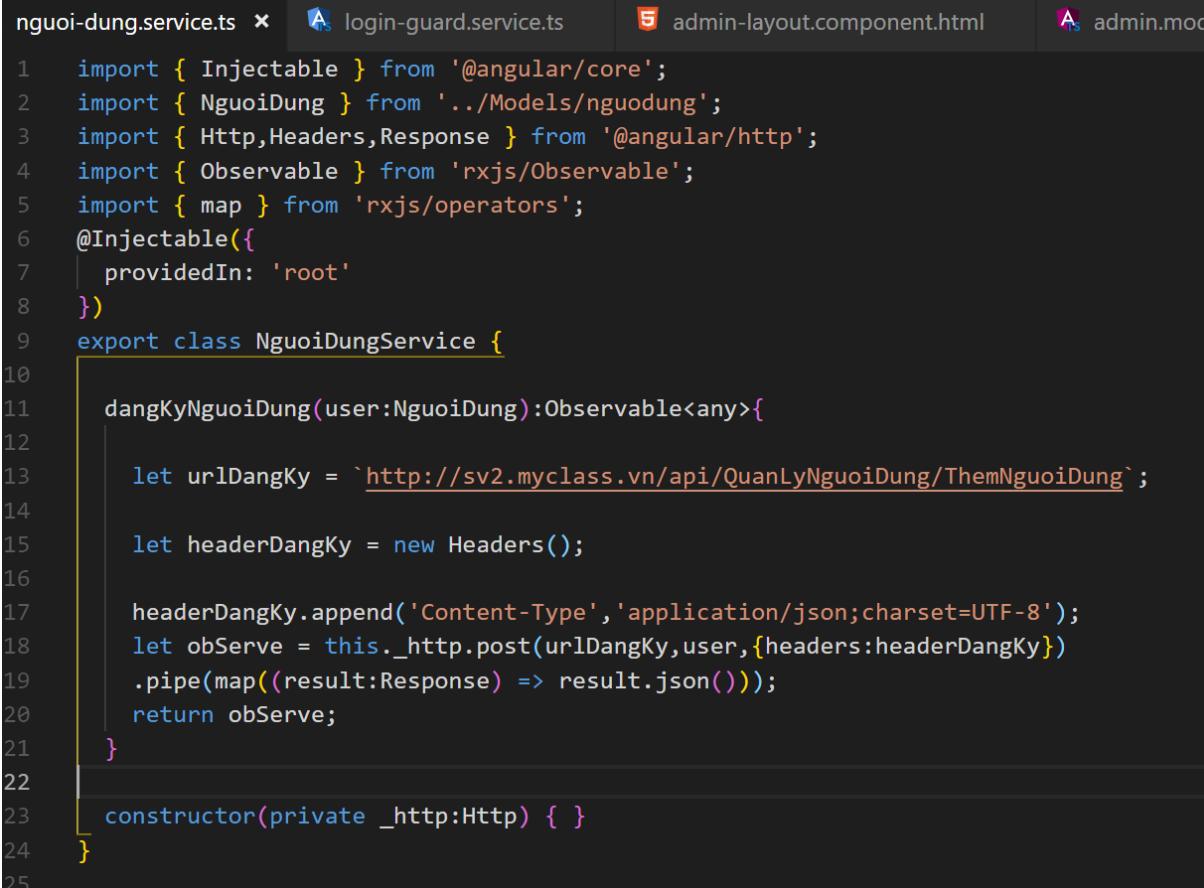


The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "PHIMPROJECT". It includes files like "tin-tuc.component.ts", "trangchitiet", "trangchu", "home.module.spec.ts", "home.module.ts", and a "Models" folder containing "lichchieu.ts", "nguodung.ts", and "Rap.ts".
- QUAN-LY-PHIM.COMPONENT.TS**: This tab is currently active, showing the component template code.
- PHIM.SERVICE.TS**: Another tab showing service logic.
- NGUODUNG.TS**: The active code editor tab, displaying the following TypeScript code for the **NguoiDung** class:

```
1 export class NguoiDung{  
2     public TaiKhoan:string;  
3     public MatKhau:string;  
4     public Email:string;  
5     public SoDT:string;  
6     public MaNhom:string;  
7     public MaLoaiNguoiDung:string;  
8     public HoTen:string;  
9 }
```

B2: Tạo class **NguoiDungService** (trong thư mục Service) Chứa các phương thức get post put delete ... Để giao tiếp với backend.



```
1 import { Injectable } from '@angular/core';
2 import { NguoiDung } from '../Models/nguodung';
3 import { Http, Headers, Response } from '@angular/http';
4 import { Observable } from 'rxjs/Observable';
5 import { map } from 'rxjs/operators';
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class NguoiDungService {
10
11   dangKyNguoiDung(user:NguoiDung):Observable<any>{
12
13     let urlDangKy = `http://sv2.myclass.vn/api/QuanLyNguoiDung/ThemNguoiDung`;
14
15     let headerDangKy = new Headers();
16
17     headerDangKy.append('Content-Type', 'application/json;charset=UTF-8');
18     let obServe = this._http.post(urlDangKy,user,{headers:headerDangKy})
19       .pipe(map((result:Response) => result.json()));
20     return obServe;
21   }
22
23   constructor(private _http:Http) { }
```

Trong đó:

- ❖ Headers: lớp hỗ trợ khai báo các metadata trong gói tin Http
- ❖ Để post được json lên phải có body và header, do backend quy định, ở đây body chính là đối tượng user để gửi lên server.
- ❖ Rxjs: thư viện hỗ trợ các tác vụ bất đồng bộ
- ❖ Response: Kiểu dữ liệu server trả về
- ❖ Map: phương thức của thư viện Rxjs để chuyển dữ liệu từ kiểu Response ra kiểu json để sử dụng

❖ Nội dung form đăng ký, sử dụng ngForm để handle form.

```
<div class="container">
<div class="row">
<div class="col-5 mx-auto">
<form #formDangKy="ngForm">
<h4 class="display-4">Đăng Ký</h4>
<div class="form-group">
<label for="">Tài Khoản</label>
<input type="text" class="form-control" ngModel>
</div>
<div class="form-group">
<label for="">Mật Khâu</label>
<input type="text" class="form-control" ngModel>
</div>
<div class="form-group">
<label for="">Họ Tên: </label>
<input type="email" class="form-control" ngModel>
</div>
<div class="form-group">
<label for="">Email</label>
<input type="email" class="form-control" ngModel>
</div>
<div class="form-group">
<label for="">Số ĐT</label>
<input type="text" class="form-control" ngModel>
</div>
<div class="form-group">
<label for="">Mã Nhóm: </label>
<select name="MaNhom" class="form-control" ngModel id="">
<option *ngFor="let nhom of mangNhom" value={{nhom}}>{{nhom}}</option>
</select>
</div>
<div class="form-group">
<label for="">Loại Người Dùng: </label>
<input type="radio" ngModel name="MaLoaiNguoiDung" value="KhachHang">Khách Hàng
<input type="radio" ngModel name="MaLoaiNguoiDung" value="QuanTri">Admin
</div>
<div class="form-group text-center">
<button type="submit" class="btn btn-success">Đăng Ký</button>
</div>
</form>
</div>
</div>
</div>
```

Nội dung typescript component Form Register

```
selector: 'app-dangky',
templateUrl: './dangky.component.html',
styleUrls: ['./dangky.component.css']
})
export class DangkyComponent implements OnInit, OnDestroy{
  @ViewChild('formDangKy') formDK: NgForm;
  mangNhóm:string[] = ['GP01','GP02','GP03','GP04','GP05','GP06','GP07','GP08','GP09'];
  mangNguoiDung:any[] = [];
  ObserveLayDSND:Subscription;

  constructor(private nguoiDungSV:NguoiDungService) { } ← Inject service vào hàm khởi tạo

  DangKy(value:NguoiDung){
    console.log(value)
    this.nguoiDungSV.dangKyNguoiDung(value).subscribe((kq:any) =>{ ↑
      console.log(kq);
      this.mangNguoiDung.push(value);
    },error =>{
      console.log(error);
    })
  }

  ngOnInit() {
```

- Về nhóm, có 10 nhóm từ “GP01” tới “GP10”, mỗi nhóm có dữ liệu phim và người dùng riêng

Đăng Ký

Tài Khoản

Mật Khẩu

Họ Tên:

Mã Nhóm:

GP03

Email

Số ĐT

Loại Người Dùng: Khách Hàng Admin

Đăng Ký

Thực hành: Lấy danh Sách phim

Thực hiện các đối tượng trên để kết nối đến server thông qua urlAPI lấy về danh sách phim
=> hiển thị ra giao diện MovieComponent.

Class Phim

```
export class Phim{
    public MaPhim;
    public TenPhim;
    public Trailer;
    public HinhAnh;
    public MoTa;
    public MaNhom;
    public NgayKhoiChieu;
    public DanhGia;
}
```

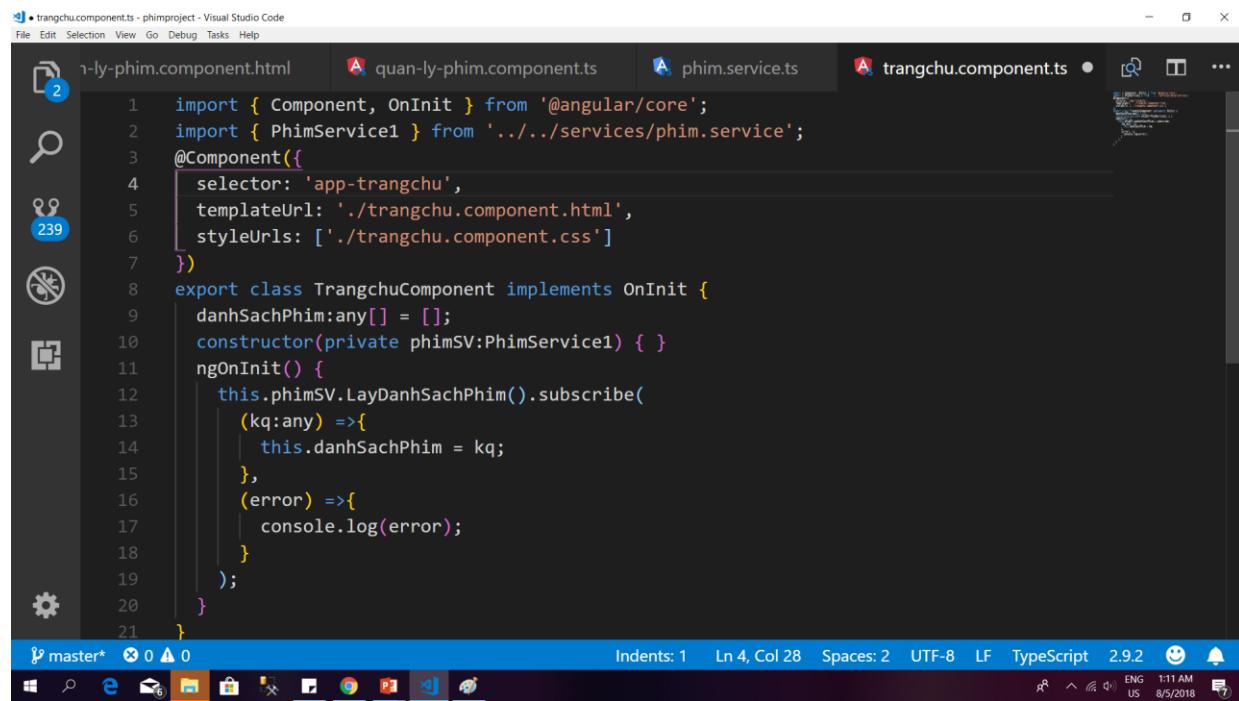
Service Phim

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

- Title Bar:** phim.service.ts - phimproject - Visual Studio Code
- File Explorer:** Shows files like jung.component.ts, quan-ly-phim.component.html, quan-ly-phim.component.ts, and phim.service.ts.
- Search Bar:** Contains the search term "LayDan".
- Code Editor:** Displays the code for the PhimService component. The code uses RxJS and HttpClient to make a GET request to a local API endpoint `http://sv2.myclass.vn/api/QuanLyPhim/LayDanhSachPhim?MaNhom=GP01` and return the result as JSON.
- Status Bar:** Indents: 2, Ln 14, Col 5, Spaces: 2, UTF-8, LF, TypeScript 2.9.2, master*, 108 AM, ENG US, 8/5/2018.

Class MovieComponent

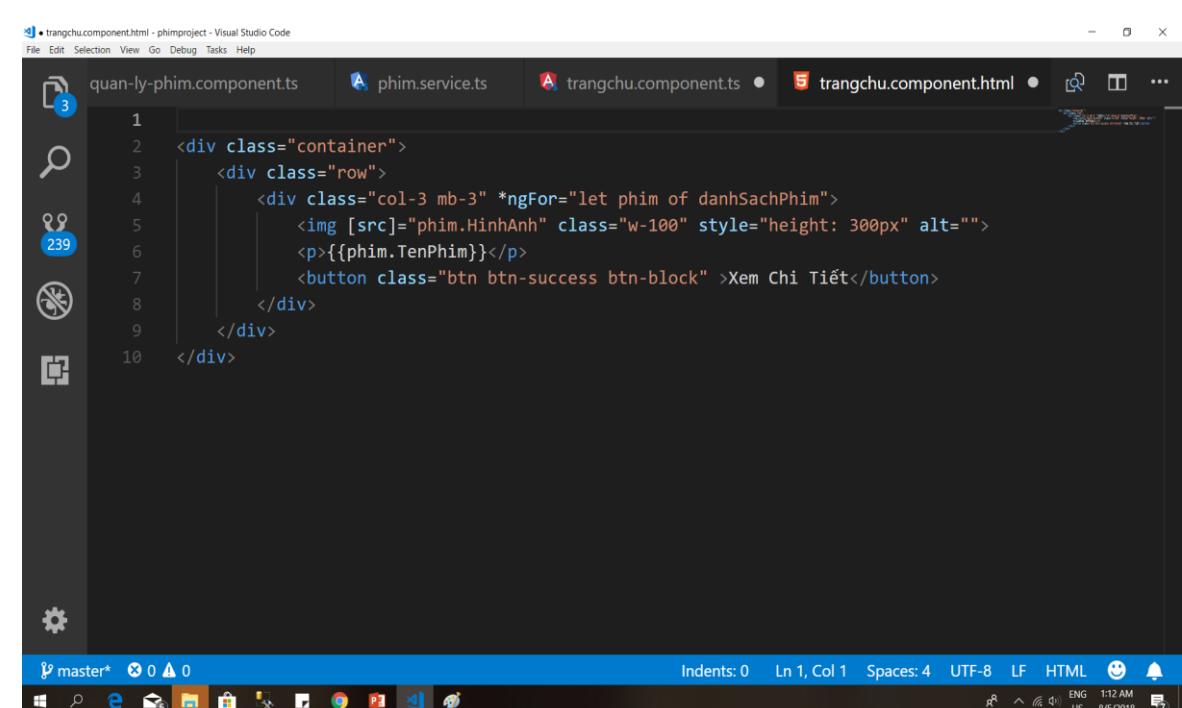
movie.component.ts



A screenshot of Visual Studio Code showing the code for `movie.component.ts`. The component uses Angular's `Component` decorator to define a selector (`'app-trangchu'`) and templateUrl (`'./trangchu.component.html'`). It injects the `PhimService1` service and implements the `OnInit` interface. The `ngOnInit` method subscribes to the `LayDanhSachPhim` observable from the service, logging the result to the console if successful or an error if there is one.

```
import { Component, OnInit } from '@angular/core';
import { PhimService1 } from '../../../../../services/phim.service';
@Component({
  selector: 'app-trangchu',
  templateUrl: './trangchu.component.html',
  styleUrls: ['./trangchu.component.css']
})
export class TrangchuComponent implements OnInit {
  danhSachPhim:any[] = [];
  constructor(private phimSV:PhimService1) { }
  ngOnInit() {
    this.phimSV.LayDanhSachPhim().subscribe(
      (kq:any) =>{
        this.danhSachPhim = kq;
      },
      (error) =>{
        console.log(error);
      }
    );
  }
}
```

movie.component.html



A screenshot of Visual Studio Code showing the code for `movie.component.html`. The template uses Angular's `ngFor` directive to iterate over the `danhSachPhim` array. For each item, it displays an image (`img` tag with `src` set to `phim.HinhAnh`), the movie title (`phim.TenPhim`), and a button labeled "Xem Chi Tiết" which is styled with `btn btn-success btn-block`.

```
<div class="container">
  <div class="row">
    <div class="col-3 mb-3" *ngFor="let phim of danhSachPhim">
      <img [src]="phim.HinhAnh" class="w-100" style="height: 300px" alt="">
      <p>{{phim.TenPhim}}</p>
      <button class="btn btn-success btn-block" >Xem Chi Tiết</button>
    </div>
  </div>
</div>
```

❖ Sử dụng service để chuyển đổi dữ liệu giữa các component

Chúng ta đã làm quen với 2 kĩ thuật :

- Input: chuyển dữ liệu từ component cha sang component con
- Output: chuyển dữ liệu từ component con sang component cha

Vậy trong trường hợp 2 component không liên quan tới nhau thì làm sao để làm??



Sử dụng service với đối tượng BehaviorSubject do angular cung cấp để chuyển đổi dữ liệu

❖ Sử dụng service để chuyển đổi dữ liệu giữa các component

B1: tạo 1 service , đặt tên là TransformService, định dạng nội dung cho nó

```
1 import { Injectable } from '@angular/core';
2 import { BehaviorSubject } from 'rxjs';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class TransformDataService {
8   private data = new BehaviorSubject(null);
9   public asData = this.data.asObservable();
10 constructor() { }
11 public transformData(thamso){
12   this.data.next(thamso);
13 }
14 }
15
```

Trong đó:

data: đối tượng **BehaviorSubject**,
được xem như trung gian chứa dữ liệu
chuyển đổi giữa các component, với
null là giá trị mặc định ban đầu đối
tương **data** đang chứa

asData: đại diện cho đối tượng data
dưới dạng observable, từ đó ta có thể
subscribe .

❖ Sử dụng service để chuyển đổi dữ liệu giữa các component

B2: tạo component AdminLayoutComponent,

B3: tạo 2 component SidebarComponent và DanhSachPhimComponent nằm trong AdminLayoutComponent

Danh sach phim.component.html

```
1 <p>
2   danh-sach-phim works!
3 </p>
4 <button class="btn btn-outline-success" (click)="toggleSidebar()">Toggle</button>
```

Danh sach phim.component.ts

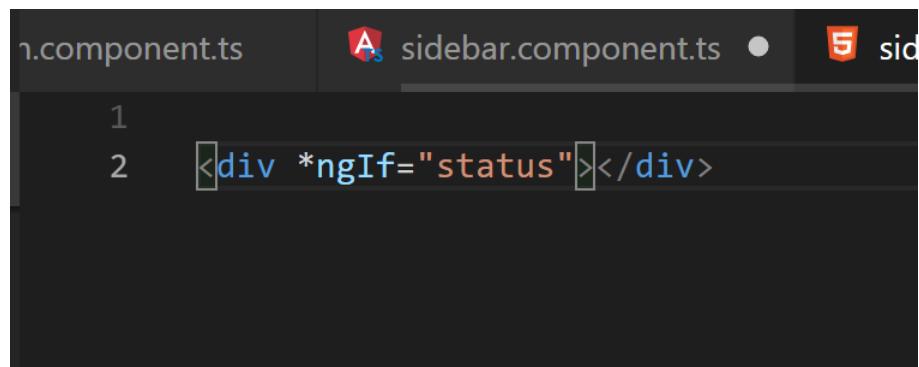
```
1 import { Component, OnInit } from '@angular/core';
2 import { TransformDataService } from '../../../../../services/transform-data.service';
3
4 @Component({
5   selector: 'app-danh-sach-phim',
6   templateUrl: './danh-sach-phim.component.html',
7   styleUrls: ['./danh-sach-phim.component.css']
8 })
9 export class DanhSachPhimComponent implements OnInit {
10   statusSideBar:boolean = false;
11
12   constructor(private transform:TransformDataService) { }
13
14   toggleSidebar(){
15     this.statusSideBar == false ? this.statusSideBar = true: this.statusSideBar = false;
16     this.transform.transformData(this.statusSideBar);
17   }
18   ngOnInit() {
19   }
20 }
```

❖ Sử dụng service để chuyển đổi dữ liệu giữa các component

B2: tạo component AdminLayoutComponent,

B3: tạo 2 component SidebarComponent và DanhSachPhimComponent nằm trong AdminLayoutComponent

sidebar.component.html



```
1
2  <div *ngIf="status"></div>
```

sidebar.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TransformDataService } from '../../../../../services/transform-data.service';
@Component({
  selector: 'app-sidebar',
  templateUrl: './sidebar.component.html',
  styleUrls: ['./sidebar.component.css']
})
export class SidebarComponent implements OnInit {
  constructor(private transform:TransformDataService) { }
  status:boolean = false;
  ngOnInit() {
    this.transform.asData.subscribe(
      (kq) =>{
        this.status = kq;
        console.log(this.status)
      }
    )
  }
}
```

❖ Sử dụng service để chuyển đổi dữ liệu giữa các component

- ☐ Ở sidebar, chúng ta có 1 div, div này sẽ hiển thị theo biến status, status= true thì hiện, status = false thì ẩn
- ☐ Từ DanhSachPhim, khi nhấn button Toggle, thì div của sidebar sẽ ẩn hiện theo, tức là khi ta nhấn nút toggle, ta sẽ phải chuyển giá trị của biến statusSideBar qua bên sideBar component và gán vào biến status.
- ☐ Ở SidebarComponent , ban đầu gán biến status = kq trả về từ asData, kq chính là giá trị mặc định ban đầu của data, là null
- ☐ Ở DanhSachPhim, nhấn nút button, ta gọi phương thức TransformData() , truyền vào tham số là biến statusSideBar, bây giờ TransformData gọi phương thức data.next(statusSideBar), từ đó, null sẽ biến thành statusSideBar => kq = statusSideBar => status = statusSideBar, tức là chúng ta chuyển dữ liệu thành công

DanhSachGhe Component

```
1 import { Component, OnInit } from '@angular/core';
2 import { TransformDataService } from '../../../../../services/transform-data.service';
3
4 @Component({
5   selector: 'app-danh-sach-phim',
6   templateUrl: './danh-sach-phim.component.html',
7   styleUrls: ['./danh-sach-phim.component.css']
8 })
9 export class DanhSachPhimComponent implements OnInit {
10   statusSideBar:boolean = false;
11
12   constructor(private transform:TransformDataService) { }
13
14   toggleSidebar(){
15     this.statusSideBar == false ? this.statusSideBar = true: this.statusSideBar = false;
16     this.transform.transformData(this.statusSideBar);
17   }
18   ngOnInit() {
19   }
20 }
```

TransformDataService

```
1 import { Injectable } from '@angular/core';
2 import { BehaviorSubject } from 'rxjs';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class TransformDataService {
8   private data = new BehaviorSubject(null);
9   public asData = this.data.asObservable();
10  constructor() { }
11  public transformData(thamso){
12    this.data.next(thamso);
13  }
14
15 }
```

Sidebar Component

```
import { Component, OnInit } from '@angular/core';
import { TransformDataService } from '../../../../../services/transform-data.service';
@Component({
  selector: 'app-sidebar',
  templateUrl: './sidebar.component.html',
  styleUrls: ['./sidebar.component.css']
})
export class SidebarComponent implements OnInit {
  constructor(private transform:TransformDataService) { }
  status:boolean = false;
  ngOnInit() {
    this.transform.asData.subscribe(
      (kq) =>{
        this.status = kq;
        console.log(this.status)
      }
    )
  }
}
```

Routing

Routing trong angular 2 là cơ chế chuyển đổi qua lại giữa các component từ đồng thời tạo ra các url cho component đó => Đây là 1 kỹ thuật giúp vừa load nhanh các thành phần của trang và đảm bảo được url (tốt cho SEO google có thể index được).

Nội dung:

1. Cấu hình route -> Định nghĩa link
2. Chuyển hướng trang

3. Tạo 2 component hoặc tái sử dụng các component trong module cũ.

Ví dụ: Sử dụng **HomeComponent** từ home module

Và **MovieComponent** trong movie module

4. Ta xem 2 component đó là 2 component lớn ở cấp độ là 1 trang.

5. Sau đó ta tạo 1 Folder RouteConfig (Folder này quản lý sự chuyển đổi qua lại giữa các component lớn (trang)

TS app.routes.ts x

```
1 import {Routes, RouterModule} from '@angular/router'
2 import { HomeComponent } from './home/home.component'
3 import { MovieComponent } from './movie/movie/movie.component';
4
5 const routing: Routes = [
6   {path: '', component: HomeComponent},
7   {path: 'movies', component: MovieComponent}
8 ];
9 export const appRoutes = RouterModule.forRoot(routing)
```

Thư viện routing

path (url): Đường dẫn đến component đó.
Component: là component sẽ được đưa vào vị trí <router-outlet>

↑
export module appRoutes này ra ngoài để import vào app.module.ts
áp dụng cho toàn hệ thống.

6. Tại app.component.ts ta import vào:

- ✓ **appRoutes** (file app.routes.ts vừa cấu hình)
- ✓ **RouterModule**(@angular/router): Module giúp cấu hình routerLink

```
TS app.module.ts ✘

17 //Router module chuyển hướng các trang
18 import { RouterModule } from '@angular/router';
19 import {appRoutes} from './RouteConfig/app.routes';
20 @NgModule({
21   declarations: [
22     AppComponent
23   ],
24   imports: [
25     BrowserModule,HomeModule, LayoutModule,DatabindingModule,DirectiveModule,PipeModule,
26     ,TestserviceModule,MovieModule,HttpModule,UserModule,appRoutes,RouterModule
27   ],
28   providers: [],
29   bootstrap: [AppComponent]
30 })
31 export class AppModule { }
```

Kết quả khi bấm vào routerlink

Trang chủ

A screenshot of a web browser window titled "Phim". The address bar shows "localhost:4200". The page content includes a header with navigation links to "Trang chủ" and "Danh sách phim", and a footer section.

Header

- [Trang chủ](#)
- [Danh sách phim](#)

home

Footer

Trang DanhSachPhim

Header

- [Trang chủ](#)
- [Danh sách phim](#)



Minions



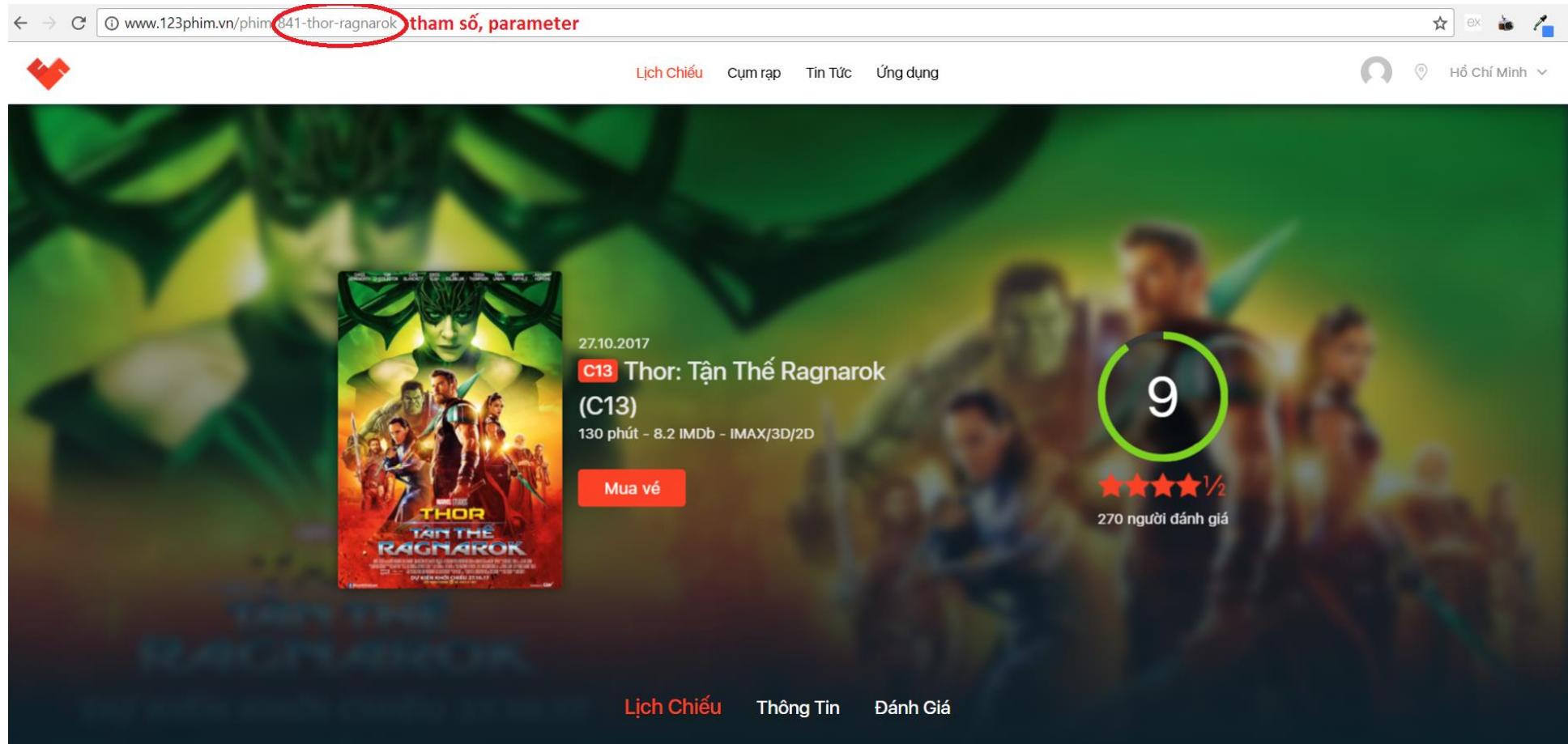
Ted 2



Trainwreck

Parameter (tham số trên URL)

Parameter dùng để truyền nhận giá trị thông qua link (Url). Từ đó ta có thể dựa trên giá trị tham số đó để truy vấn trích xuất dữ liệu.



Có 2 trường hợp truyền nhận dữ liệu thông qua link (routerlink) và sự kiện.

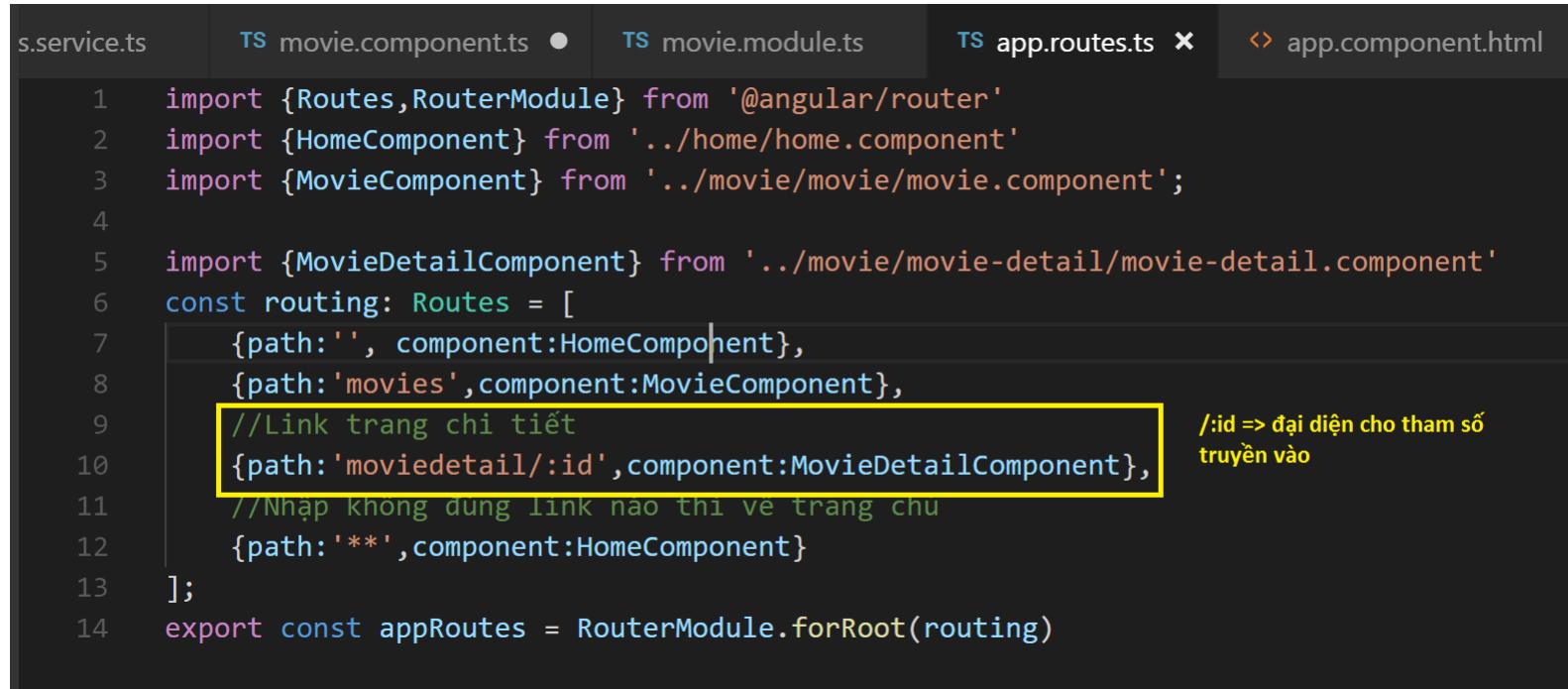
- ❖ Truyền nhận 1 tham số
- ❖ Truyền nhận nhiều tham số

❖ Truyền 1 tham số

Thực hành:

B1: Tạo 1 **MovieDetailComponent** (Đại diện trang chi tiết) ->tạo trong **MovieModule** luôn

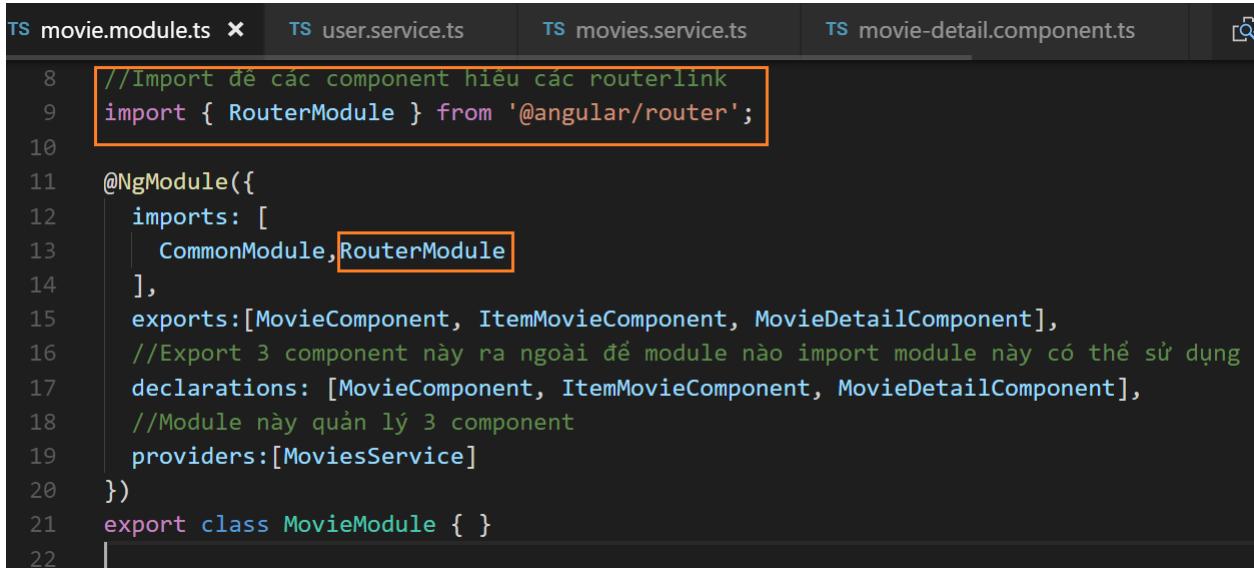
B2: Tại **app.routes.ts** thêm vào đoạn code phía dưới. Đoạn có đó định nghĩa là đường link đó + với tham số. (**path: ‘moviedetail/:id’**)



```
s.service.ts      TS movie.component.ts ●   TS movie.module.ts      TS app.routes.ts ✘   ◊ app.component.html
1 import {Routes, RouterModule} from '@angular/router'
2 import {HomeComponent} from '../home/home.component'
3 import {MovieComponent} from '../movie/movie/movie.component';
4
5 import {MovieDetailComponent} from '../movie/movie-detail/movie-detail.component'
6 const routing: Routes = [
7   {path: '', component: HomeComponent},
8   {path: 'movies', component: MovieComponent},
9   //Link trang chi tiết
10  [path: 'moviedetail/:id', component: MovieDetailComponent],
11  //Nhập không đúng link nào thì về trang chủ
12  {path: '**', component: HomeComponent}
13 ];
14 export const appRoutes = RouterModule.forRoot(routing)
```

:id => đại diện cho tham số
truyền vào

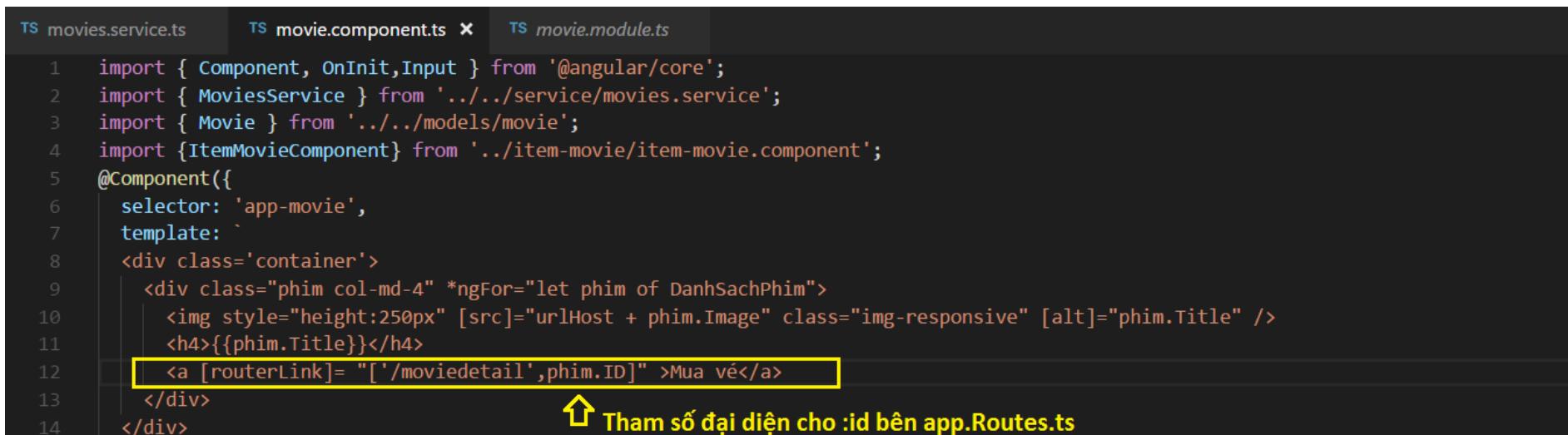
B3: Tại MovieModule import RouterModule (của @angular/router)



```
TS movie.module.ts × TS user.service.ts TS movies.service.ts TS movie-detail.component.ts
8 //Import để các component hiểu các routerLink
9 import { RouterModule } from '@angular/router';
10
11 @NgModule({
12   imports: [
13     CommonModule, RouterModule
14   ],
15   exports:[MovieComponent, ItemMovieComponent, MovieDetailComponent],
16   //Export 3 component này ra ngoài để module nào import module này có thể sử dụng 3
17   declarations: [MovieComponent, ItemMovieComponent, MovieDetailComponent],
18   //Module này quản lý 3 component
19   providers:[MoviesService]
20 })
21 export class MovieModule { }
22 |
```

Tại **MovieModule** ta truyền vào **RouterModule** để **componentMovie** hiểu được thuộc tính **[routerLink]**

Tại **MovieComponent** ta truyền vào 1 đường link



```
TS movies.service.ts × TS movie.component.ts × TS movie.module.ts
1 import { Component, OnInit,Input } from '@angular/core';
2 import { MoviesService } from '../../service/movies.service';
3 import { Movie } from '../../../../../models/movie';
4 import {ItemMovieComponent} from '../item-movie/item-movie.component';
5 @Component({
6   selector: 'app-movie',
7   template: `
8     <div class='container'>
9       <div class="phim col-md-4" *ngFor="let phim of DanhSachPhim">
10         <img style="height:250px" [src]="urlHost + phim.Image" class="img-responsive" [alt]="phim.Title" />
11         <h4>{{phim.Title}}</h4>
12         <a [routerLink]= "[\"/moviedetail\",phim.ID]" >Mua vé</a>
13       </div>
14     </div>
15   </div>
```

↑ Tham số đại diện cho :id bên app.Routes.ts

B4: Tại **MovieDetailComponent** ta Import 2 thư viện để lấy được tham số trên URL

ActivatedRoute, Router: Dùng để lấy nội dung url.

Subscription: Để đăng ký theo dõi tham số lấy từ url cũng như service

Để lấy được giá trị tham số ta cần sử dụng 2 đối tượng
ActivatedRoute và Subscription

html load nội dung

```
1 import { Component, OnInit, OnDestroy } from '@angular/core'
2 import { [ActivatedRoute, Router] } from '@angular/router';
3 import { Subscription } from 'rxjs';
4 import { MoviesService } from '../../../../../service/movies.service';
5
6 @Component({
7   selector: 'app-movie-detail',
8   template: <div class="container">
9     <div class="col-md-3" >
10       <img style="height:250px" [src]="urlHost + MovieDetail.Image" class="img-responsive" [alt]="MovieDetail.Title" />
11     </div>
12     <div class="col-md-9">
13       <table class="table">
14         <tr><th>Tên phim</th><td>{{MovieDetail.Title}}</td></tr>
15         <tr><th>...</th><td>...</td></tr>
16       </table>
17     </div>
18     <div class="col-md-12">
19       Lịch chiếu ...
20     </div>
21   </div>
22   styleUrls: ['./movie-detail.component.css']
23 })
```

B5:

- ✓ Cũng tại **MovieDetailComponent** phần code **typescript** ta khởi tạo đối tượng **ActivatedRoute** và **Router** tại **constructor**. Để sử dụng lấy tham số từ url.
- ✓ Sau đó dùng **tham số** đó để gọi service lấy dữ liệu của chi tiết phim đó. (Ta gọi hàm này trong **OnInit()** để đảm bảo khi người dùng **bấm vào link** hoặc **nhập vào url** dữ liệu sẽ được load lên liền ngay lập tức.

```
export class MovieDetailComponent implements OnInit {  
    private urlHost:string = 'http://sv.myclass.vn/Images/Movies/';  
    private MovieDetail:any = {};  
    private MovieID:number;  
    private MaNhom:any;  
    constructor(private router:Router,private activatedRoute:ActivatedRoute,private movieService:MoviesService) { }  
    ngOnInit() {  
        //Lấy chi tiết phim lịch chiếu dựa vào 1 tham số id  
        this[activatedRoute] params.subscribe(thamso=>{  
            this.MovieID = thamso['id'];  
        });  
        this.movieService.LayChiTietPhim_LichChieu(this.MovieID).subscribe((result:any)=>{  
            this.MovieDetail = result;  
        },error => {  
            this.MovieDetail = error;  
        });  
    }  
}
```

Thông qua **activetedRoute** ta dùng phương thức **subscribe** để cắt lấy phần tham số trên **linkURL** sau đó gán vào thuộc tính **MovieID** để gọi service.

B6:

- ✓ Ta dùng 2 đối tượng pramsSub1 và Paramssub2 để bắt giá trị từ activatedroutes.params về (vì sao vì activatedroutes.params trả về là 1 đối tượng observable cần 1 đối tượng để hóng lấy giá trị đúng mờ dữ liệu khi có thể).
- ✓ Component này kế thừa từ OnDestroy để hủy theo dõi biến từ observable trả về giải phóng biến.

```
export class MovieDetailComponent implements OnInit, OnDestroy {
  private urlHost: string = 'http://sv.myclass.vn/Images/Movies/';
  private MovieDetail: any = {}; //Đối tượng movie
  private MovieID: number;
  private MaNhom: any;
  private paramsSub1: Subscription; //Là đối tượng chứa kết quả có được sau khi thực hiện một observable, nó thường dùng cho việc hủy hoặc tiếp tục xử lý.
  private paramsSub2: Subscription;
  constructor(private router: Router, private activatedRoute: ActivatedRoute, private movieService: MoviesService) {}

  ngOnInit() {
    //Lấy chi tiết phim lịch chiếu dựa vào 1 tham số id
    this.paramsSub1 = this.activatedRoute.params.subscribe(thamso=>{
      this.MovieID = thamso['id'];
    });
    this.paramsSub2 = this.movieService.LayChiTietPhim_LichChieu(this.MovieID).subscribe((result:any)=>{
      this.MovieDetail = result;
    },error => {
      this.MovieDetail = error;
    });
  }

  ngOnDestroy(){
    this.paramsSub1.unsubscribe(); //Phương thức hủy đăng ký theo dõi giá trị biến, khi ta
    this.paramsSub2.unsubscribe(); //đã lấy được giá trị vào thuộc tính cần thiết
  }
}
```

Là đối tượng chứa kết quả có được sau khi thực hiện một observable, nó thường dùng cho việc hủy hoặc tiếp tục xử lý.

Phương thức thuộc đối tượng observable dùng để (mở đường ống) theo dõi giá trị biến trả về từ params

activatedRoute.params trả về đối tượng là observable

B7: Tại class **MovieService** xây dựng thêm phương thức lấy dữ liệu phim từ server thông qua tham số **MaPhim**.

Chú ý:

id=\${MaPhim} đại diện cho phần ta vừa cấu hình bên **app.routes.ts**

```
//Link trỏ đến api backend lấy thông tin phim, lịch chiếu, ...
private apiUrlGetMovieDetail = 'http://sv.myclass.vn/api/movie/GetMovieDetail';
//Lấy thông tin chi tiết của 1 phim dựa trên mã phim
public LayChiTietPhim_LichChieu(MaPhim:number):any {
  //map(): để đưa kiểu dữ liệu 1 chuỗi về dạng đối tượng response => từ reponse ta có thể parse ra kiểu dữ liệu json
  let obServe:Observable<any> = this._http.get(`${this.apiUrlGetMovieDetail}?id=${MaPhim}`).map((result:Response) => result.json());
  return obServe;
}
```

Xây dựng 1 phương thức service lấy chi tiết của phim dựa vào
mã phim (Link server backend quy định)

❖ Kết quả

Từ trang **Movies**(DanhSachPhim) các item chứa các router link + MovieID

Header

- Trang chủ
- Danh sách phim

The screenshot shows a list of movies with their posters and titles. A red arrow points from the text below to the 'Minions' card. The 'Mua vé' button for 'Minions' is highlighted with a red box.

router linh + tham số MovieID khi click vào chuyển đến component moviedetail

Minions Mua vé

Ted 2

Trainwreck

Mua vé

Trang **MovieDetail** lấy được tham số gọi service lấy dữ liệu => load dữ liệu phim lịch chiếu ra

Header

- Trang chủ
- Danh sách phim

The screenshot shows the details for the 'Minions' movie. It includes the movie poster, title 'Tên phim' (Movie Name), and a section for 'Lịch chiếu ...' (Movie Showtimes). The 'Mua vé' button is visible at the bottom.

Tên phim

Minions

...

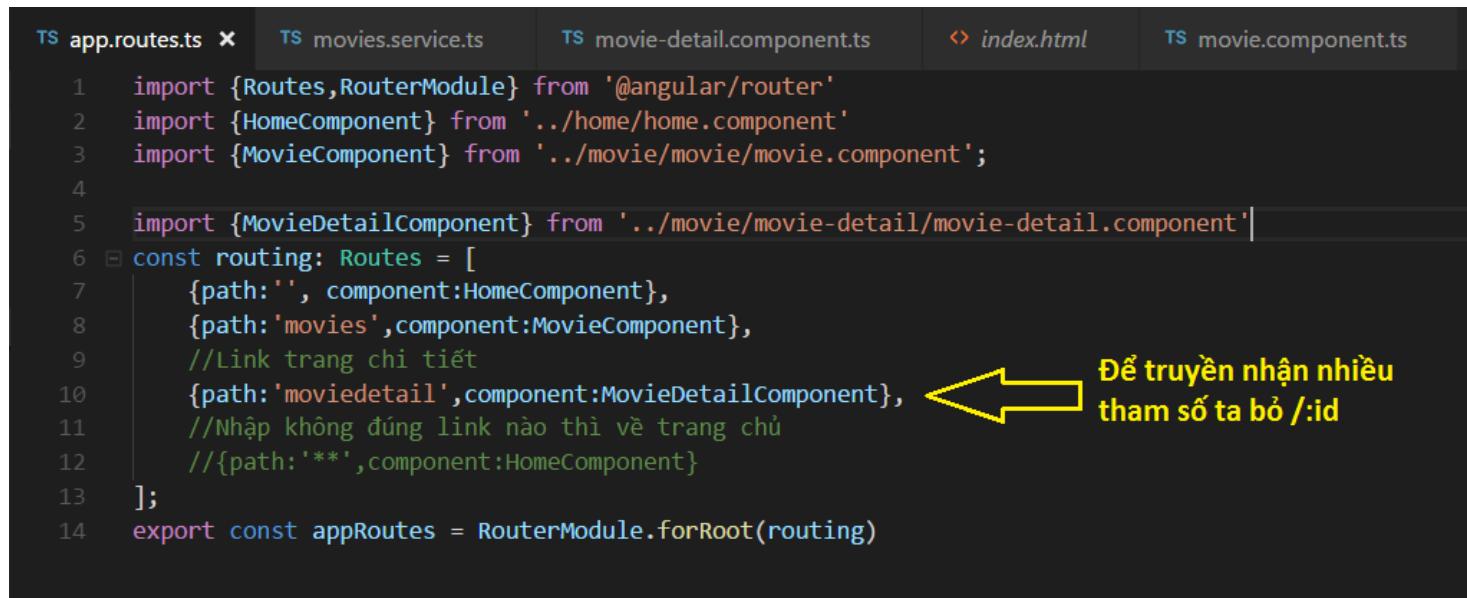
Lịch chiếu ...

Mua vé

Footer

❖ Ví dụ về truyền nhận nhiều tham số

App.routes.ts



```
TS app.routes.ts x TS movies.service.ts TS movie-detail.component.ts <> index.html TS movie.component.ts
1 import {Routes, RouterModule} from '@angular/router'
2 import {HomeComponent} from '../home/home.component'
3 import {MovieComponent} from '../movie/movie/movie.component';
4
5 import {MovieDetailComponent} from '../movie/movie-detail/movie-detail.component'
6 const routing: Routes = [
7   {path: '', component: HomeComponent},
8   {path: 'movies', component: MovieComponent},
9   //Link trang chi tiết
10  {path: 'moviedetail', component: MovieDetailComponent}, //Nhập không đúng link nào thì về trang chủ
11  //{path: '**', component: HomeComponent}
12 ];
13
14 export const appRoutes = RouterModule.forRoot(routing)
```

Để truyền nhận nhiều
tham số ta bỏ /:id

MoviesComponent (DanhSachPhim)

```
TS app.routes.ts    TS movies.service.ts    TS movie-detail.component.ts    index.html    TS movie.component.ts x
1 import { Component, OnInit, Input } from '@angular/core';
2 import { MoviesService } from '../../../../../service/movies.service';
3 import { Movie } from '../../../../../models/movie';
4
5 @Component({
6   selector: 'app-movie',
7   template: `
8     <div class='container'>
9       <div class="phim col-md-4" *ngFor="let phim of DanhSachPhim">
10         <img style="height:250px" [src]="urlHost + phim.Image" class="img-responsive" [alt]="phim.Title" />
11         <h4>{{phim.Title}}</h4>
12         <a [routerLink]= "[ '/moviedetail',phim.ID]" >Mua vé</a><br>
13         <a [routerLink]= "[ '/moviedetail'" [queryParams] = " {id:phim.ID,groupID:MaNhom}">Mua vé (GroupID: {{MaNhom}}) </a>
14       </div>
15     </div>
16   `,
17   styleUrls: ['./movie.component.css']
18 })
19 export class MovieComponent implements OnInit {
20   private MaNhom:string = 'GP01';
21   //LINK trỏ đến thư mục hình ảnh từ server
22   private urlHost:string = 'http://sv.myclass.vn/Images/Movies/';
23   //Đối tượng nhận dữ liệu trả về khi gọi đến server
24   private DanhSachPhim:Array<Movie>;
25   constructor(private servicePhim:MoviesService) {
26     //Constructor được gọi trước khi component khởi tạo.
27   }
28   ngOnInit() {
29     //ngOnInit: Được gọi đầu tiên sau khi component khởi tạo xong.
30     this.servicePhim.LayDanhSachPhim().subscribe((result:Array<Movie>)=>{
31       this.DanhSachPhim = result;
32     },error => {
33       this.DanhSachPhim = error;
34     });
35   }
36 }
37
```

Bảng tab hiển thị các tệp tin:

- app.routes.ts
- movies.service.ts
- movie-detail.component.ts
- index.html
- movie.component.ts

Trong file movie.component.ts:

- Đường dẫn routerLink để chuyển sang chi tiết phim: `[routerLink= "['/moviedetail',phim.ID]"]`
- Đường dẫn routerLink để mua vé: `[routerLink= "['/moviedetail'" [queryParams] = " {id:phim.ID,groupID:MaNhom}"]`
- Biến `MaNhom` có giá trị `'GP01'`.
- Biến `urlHost` có giá trị `'http://sv.myclass.vn/Images/Movies/'`.
- Biến `DanhSachPhim` là một mảng của type `Movie`.
- Constructor `constructor(private servicePhim:MoviesService)` để khai báo khung.
- Method `ngOnInit()` để khai báo khung.

Điều đặc biệt:

- Đường dẫn routerLink cho mua vé bao gồm `[queryParams] = " {id:phim.ID,groupID:MaNhom}"`. Một chú thích màu xanh lá cây giải thích rằng `Tham số được truyền thông qua properties binding queryParams`.
- Một chú thích màu vàng giải thích rằng `Bảng tham số chỉ truyền path` (nhỏ) và `Tham số được truyền thông qua properties binding queryParams` (lớn).

MoviesComponentDetail (Chi tiết + lịch chiếu)

```
ts app.routes.ts      ts movies.service.ts    ts movie-detail.component.ts    index.html    ts movie.component.ts x
export class MovieDetailComponent implements OnInit {
  private urlHost:string = 'http://sv.myclass.vn/Images/Movies/';
  private MovieDetail:any = {} //Đối tượng movie
  private MovieID:number;
  private MaNhom:any;
  private DanhSachThamSo:any
  constructor(private router:Router,private activatedRoute:ActivatedRoute,private movieService:MoviesService ) { }
  ngOnInit() {
    //Lấy chi tiết phim lịch chiếu dựa vào 1 tham số id
    // this.DanhSachThamSo = this.activatedRoute.params.subscribe(thamso=>{
    //   this.MovieID = thamso['id'];
    // });
    // this.movieService.LayChiTietPhim_LichChieu(this.MovieID).subscribe((result:any)=>{
    //   this.MovieDetail = result;
    // },error => {
    //   this.MovieDetail = error;
    // });
    //Lấy chi tiết phim lịch chiếu dựa vào nhiều hơn 1 tham số
    this.activatedRoute.queryParams.subscribe(thamso=>{
      this.MovieID = parseInt(thamso['id']);
      this.MaNhom = thamso['groupid'];
    });
    this.movieService.LayChiTietPhim_LichChieuTheoNhom(this.MovieID,this.MaNhom).subscribe((result:any)=>{
      this.MovieDetail = result;
    },error => {
      this.MovieDetail = error;
    });
  }
}
```

Thay vì **.params()** ta
.queryParams() rồi lấy
tham số gọi service

- ❖ Tương tự ta cũng có thể chuyển hướng trang hay truyền tham số thông qua sự kiện

appRouters

```
const appRouters:Routes = [
  {path:'header', component:HeaderComponent},
  {path:'login',component:LoginComponent},
  {path:'productdetail/:id',component:ProductDetailComponent},
```

Thông qua sự kiện (ở đây đặt thẻ button và sự kiện (click)

```
TruyenNieuThamSo(){
  //Truyền nhiều tham số ứng với queryparam
  this.router.navigate(['/productdetail'],{queryParams:{id:"product1",name:"nam1"}});
}

Truyen1ThamSo(){
  //Truyền 1 tham số
  this.router.navigate(['/productdetail','1']);
}
```

❖ Mặc định khi đường dẫn không có trong appRouters

Trong: const *appRouters* ta cấu hình thêm path này

```
//Ta có component dành cho các trang không xác định để về component này  
//PageNotFound Component  
{  
  path: "**", component:PageNotFoundComponent}  
}
```

Path này sẽ trả về trang

(component PageNotFound khi ta nhập đường dẫn không có trong appRouters)

Hoặc ta có thể định nghĩa trước path cho component trang này

Sau đó cấu hình path ** chuyển hướng trang bằng cú pháp redirectTo như hình bên dưới

```
//Ta có component dành cho các trang không xác định để về component này  
//PageNotFound Component  
{path: 'PageNotFound', component:PageNotFoundComponent},  
[path: "**", redirectTo: '/PageNotFound', pathMatch: 'full']}
```

Thuộc tính **PathParam: 'full'**: Ý nghĩa là toàn đường dẫn không chứa dấu / trong url

17. Animation trong angular 4

- Cài đặt: **npm install animation --save @angular/animations**
- Import **BrowserAnimationModule** vào **app.module.ts**

```
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
```

- Tạo component muốn sử dụng animation thì import các thư viện

```
import {  
  trigger,  
  state,  
  style,  
  transition,  
  animation,  
  keyframes,  
  group,  
  animate  
} | from '@angular/animations';
```

Ví dụ về animation

```
@Component({
  selector: 'app-animation',
  template: `<div class="container">
    <table class='table'>
      <tr>
        <td>Movie ID</td>
        <td>Title</td>
        <td>Image</td>
      </tr>
      <tr *ngFor="let phim of DanhSachPhim"
        [HieuUngZoom]="state"
        (click)="toggleState()>
        <td>phim.MovieID</td>
        <td>phim.Title</td>
        <td><img [src]="phim.Image" /></td>
      </tr>
    </table>
  </div>
  ,
  styleUrls: ['./animation.component.css'],
  animations: [
    trigger('HieuUngZoom',[ 
      state('inactive', style({
        backgroundColor: '#eee',
        transform: 'scale(1)'
      })),
      state('active',   style({
        backgroundColor: '#cf8dc',
        transform: 'scale(1.1)'
      })),
      transition('inactive => active', animate('100ms ease-in')),
      transition('active => inactive', animate('100ms ease-out'))
    ])
]
```

@HieuUngZoom: Tên của animation
State: Biến xác định trạng thái của animation
(Hiện tại ở đây ta có 2 trạng thái là inactive và active) ứng với 2 style css phía dưới

Tại đây ta có 1 sự kiện giúp binding trạng thái. Thông qua sự kiện click làm thay đổi biến state(biến trạng thái) thông qua phương thức toggleState().
Ngoài ra ta có thể thay đổi sự kiện click này mouse over, mouse leave, key up, key down ...

```
export class AnimationComponent implements OnInit {
  public state = 'inactive';
  public DanhSachPhim: Array<any> = [
    {
      MovieID: 'MV01',
      Title: 'Spider Man',
      Image: 'https://picsum.photos/200/300'
    },
    {
      MovieID: 'MV02',
      Title: 'Infinity War',
      Image: 'https://picsum.photos/200/300'
    },
    {
      MovieID: 'MV03',
      Title: 'Wonder woman',
      Image: 'https://picsum.photos/200/300'
    }
  ]
}

constructor() { }

toggleState() {
  this.state = this.state === 'active' ? 'inactive' : 'active';
  console.log(this.state);
}

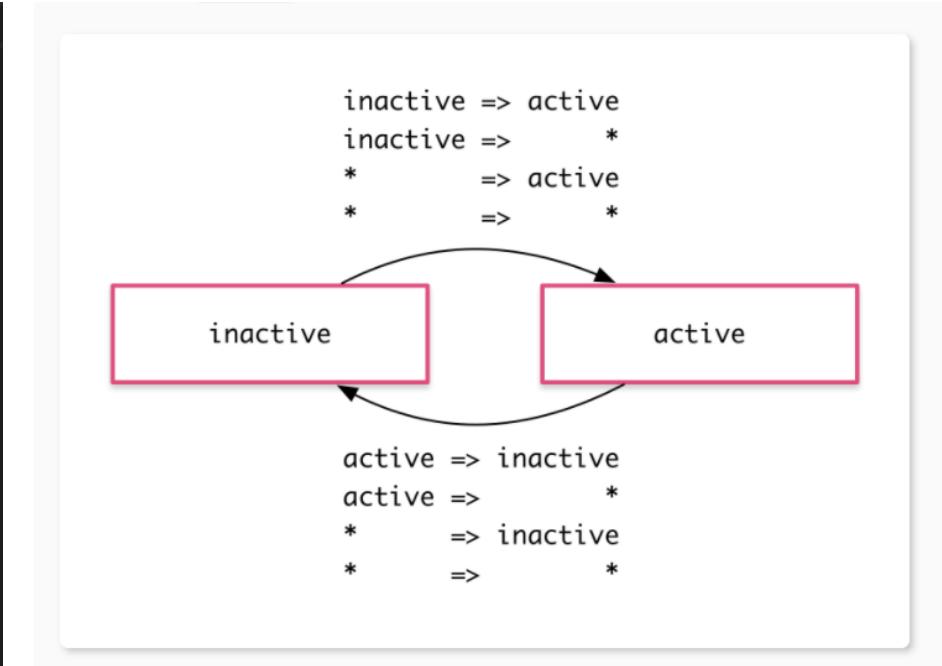
ngOnInit() {
}
```

()Ghi chú cho phần có viền màu xanh:**
Hàm xác định giá trị biến trạng thái từ giá trị nào sang giá trị nào để chuyển đổi tương ứng so với trạng thái ta định nghĩa. animate (kèm theo thời gian và hiệu ứng chuyển đổi)

Dùng *

```
animations: []
  trigger('HieuUngZoom', [
    state('inactive', style({
      backgroundColor: '#eee',
      transform: 'scale(1)'
    })),
    state('active', style({
      backgroundColor: '#cf8dc',
      transform: 'scale(1.1)',
    })),
    transition('inactive => *', animate('100ms ease-in')),
    transition('* => *', animate('100ms ease-out'))
  ])
}
```

*: Đại diện khi có sự thay đổi giá trị của biến state.
inactive => *: Từ inactive qua giá trị gì cũng sẽ áp dụng thời gian và hiệu ứng chuyển đổi đó animation(100ms ease-in)
* => *: Bất kể biến state nhận giá trị nào thì nó cũng áp dụng thời gian và hiệu ứng đó.
* => * là hiệu ứng bất kỳ sang hiệu ứng bất kỳ.



Dùng void

Trạng thái void: Dùng để định nghĩa lúc phần tử chưa xuất hiện trên giao diện.
(trạng thái void được quy định sẵn không cần định nghĩa)

Ví dụ từ:

void -> [trạng thái nào đó]: Phần tử từ khi chưa xuất hiện trên giao diện -> trạng thái css của trạng thái đích.

[trạng thái nào đó] -> void: phần tử mất dần trên giao diện

Ví dụ về void

```
@Component({
  selector: 'app-animation',
  template: `
    <div class="container">
      <table class='table'>
        <tr>
          <td>Movie ID</td>
          <td>Title</td>
          <td>Image</td>
        </tr>
        <tr *ngFor="let phim of DanhSachPhim"
            [@HieuUngLacTroi]="phim.State"
            (click)="XoaPhim(phim.MovieID)">
          <td>{{phim.MovieID}}</td>
          <td>{{phim.Title}}</td>
          <td><img [src]={{phim.Image}} /></td>
        </tr>
      </table>
      MovieID: <input #movieID /><br />
      Movie Title: <input #movieTitle /><br />
      <button (click)="ThemPhim(movieID.value,movieTitle.value)">Thêm phim </button>
    </div>
  `,
  styleUrls: ['./animation.component.css'],
  animations: [
    trigger('HieuUngLacTroi', [
      state('display', style({transform: 'translateX(0)', color: "red"})),
      transition('void => *', [
        style({transform: 'translateX(-100%)', color: 'green'}),
        animate(500)
      ]),
      transition('* => void', [
        animate(500, style({transform: 'translateX(100%)', opacity: 0}))
      ])
    ])
  ]
})
```

Ở đây ta định nghĩa duy nhất 1 trạng thái là `display` với `translateX(0)` : tại vị trí gốc của phần tử
(1): Trạng thái từ chưa xuất hiện trên giao diện là `translateX(-100%)` đến khi xuất hiện trên giao diện trong vòng 500ms và chuyển đổi màu đến vị trí `translateX(0)`: Trạng thái đích (`display`).
(2): Trạng thái bất kì -> di chuyển đến `translateX(100%)` và `opacity` giảm dần đến 0 rồi biến mất. trong khoảng thời gian là 500ms

```
export class AnimationComponent implements OnInit {
  public DanhSachPhim: Array<any> = [
    { MovieID: 'MV01',
      Title: 'Spider Man',
      Image: 'https://picsum.photos/20/30',
      State: "display"},,
    { MovieID: 'MV02',
      Title: 'Infinity War',
      Image: 'https://picsum.photos/20/30',
      State: "display"},,
    { MovieID: 'MV03',
      Title: 'Wonder woman',
      Image: 'https://picsum.photos/20/30',
      State: "display"},]
  constructor() { }
  ngOnInit() { }
```

```
XoaPhim(MovieID) {
  let i = 0;
  for(let phim of this.DanhSachPhim){
    if(phim.MovieID == MovieID)
    {
      phim.State = "void";
      this.DanhSachPhim.splice(i,1);
    }
    i++;
  }
}
```

Trước khi ta xóa phim ta xét thuộc tính trạng thái của phim thành `void` để phim vẫn còn và tiến hành remove phim khỏi đối tượng

```
ThemPhim(id,title)
{
  let phim = {
    MovieID: id,
    Title: title,
    Image: 'https://picsum.photos/20/30',
    State: "display"
  }
  this.DanhSachPhim.push(phim);
}
```

Tương tự thêm phim cũng vậy ta cho trạng thuộc tính `State` ban đầu của phim là `display`. Thi mặc định khi phim được load ra nó sẽ chuyển từ `void->display`

Định nghĩa animation kết hợp

Nếu 2 trạng thái của animation có cùng định nghĩa về thời gian và hiệu ứng ta có thể viết:

```
transition('inactive => active, active => inactive',  
          animate('100ms ease-out'))
```

Hoặc ta có thể viết:

```
transition('inactive <=> active', animate('100ms ease-out'))
```

Animation void

Nếu 2 trạng thái của animation có cùng định nghĩa về thời gian và hiệu ứng ta có thể viết:

```
transition('inactive => active, active => inactive',  
          animate('100ms ease-out'))
```

Hoặc ta có thể viết:

```
transition('inactive <=> active', animate('100ms ease-out'))
```

Animation Keyframe

Ngoài ra để chia nhỏ và customize quá trình biến đổi ta có thể dùng keyframe để định nghĩa cụ thể cho từng giai đoạn.

```
animations: [
  trigger('flyInOut', [
    state('in', style({transform: 'translateX(0)' })),
    transition('void => *', [
      animate(300, keyframes([
        style({opacity: 0, transform: 'translateX(-100%)', offset: 0}),
        style({opacity: 1, transform: 'translateX(15px)', offset: 0.3}),
        style({opacity: 1, transform: 'translateX(0)', offset: 1.0})
      ])),
      transition('* => void', [
        animate(300, keyframes([
          style({opacity: 1, transform: 'translateX(0)', offset: 0}),
          style({opacity: 1, transform: 'translateX(-15px)', offset: 0.7}),
          style({opacity: 0, transform: 'translateX(100%)', offset: 1.0})
        ]))
      ])
    ])
  ])
]
```

Animation tài liệu

Các bạn lưu ý: Khi nắm vững nguyên lý hoạt động của animation ta có thể tham khảo 1 số thư viện animation angular 4 để sử dụng và customize lại để đỡ mất thời gian cũng như hiệu ứng đẹp và chuyên nghiệp hơn. (Tóm lại là mượn code của người khác rồi sửa lại cho phù hợp với nhu cầu của mình ^_^).

Một số thư viện tham khảo:

<https://github.com/jiayihu/ng-animate>

<http://www.phpcodify.com/10-angular-2-animations-effects-tutorials/>

Cơ chế guard

Cơ chế này giúp xác nhận người dùng có thể vào được trang đó không.

Thực hành:

Tạo 1 service thuộc folder services đặt tên: **auth.guard.ts** (lưu ý phần hậu tố là .guard.ts)

```
import { Injectable } from '@angular/core';
import { CanActivate,
  ActivatedRoute,
  RouterStateSnapshot,
  Router,
  ActivatedRouteSnapshot } from '@angular/router';

@Injectable()

export class AuthGuard implements CanActivate {

  constructor(private router: Router) {}

  canActivate(activatedRoute: ActivatedRouteSnapshot, routerState: RouterStateSnapshot) {
    //Kiem tra quyền có thể gọi service đăng nhập để kiểm tra
    //Tóm lại hàm này sẽ trả về 1 giá trị là bool nếu true thì được phép và false thì không
    //Ở đây mình set localStorage đơn giản để ví dụ thôi
    if (localStorage.getItem("isAdmin") === 'admin') {
      //Nếu localStorage(isAdmin) == chuỗi admin thì dc quyền truy cập nên trả về true
      return true;
    } else {
      this.router.navigate(["PageNotFound"]); //Ngược lại load sang trang khác và return false
    }
    return false;
  }
}
```

❖ Cơ chế guard

Tại app.routes.ts

```
const appRouters:Routes = [
  {path:'home',component:AdminComponent},
  {path:'home',component:HomeComponent,children:[
    {path:'trang-chu',component:TrangchuComponent},
    {path:'trang-chitiet',component:TrangchitietComponent}
  ]},
  {path:'admin',component:AdminComponent,canActivate:[AuthGuard],children:[
    {path:'quan-ly-phim',component:QuanlyphimComponent},
    {path:'quang-ly-nguo-dung',component:QuanlynguoidungComponent}
  ]},
  {path:'PageNotFound',component:PageNotFoundComponent}
];
```