TÌM HIỂU VỀ CẤU TRÚC CỦA ANGULAR

1. Cài đặt
   1. Cài đặt ứng dụng Angular Cli

* Angular Cli được cài đặt thông qua cửa sổ lệnh command
* Cú pháp cài đặt như sau:

|  |
| --- |
| *npm install -g @angular/cli* |

* Kiểm tra phiên bản hiện tại trên máy tính thông qua câu lệnh:

|  |
| --- |
| *ng v* |

* 1. Tạo Project bằng Angular Cli
* Để tạo Project, đầu tiên ta bật cửa sổ thực thi dòng lệnh, sau đó điều hướng tới thư mục muốn tạo Project. Tiếp theo ta sử dụng cú pháp sau đây để tạo Project Angular với Angular Cli như sau:

|  |
| --- |
| *ng new PROJECT-NAME* |

*\*Với PROJECT-NAME là tên dự án muốn tạo*

## 1.3 Chạy ứng dụng

|  |
| --- |
| *cd PROJECT-NAME*  *ng serve --open* |

* Lệnh ng serve khởi chạy máy chủ, xem các tệp của chúng ta và xây dựng lại ứng dụng khi chúng ta thực hiện các thay đổi đối với các tệp đó.
* Tùy chọn --open (hoặc -o) tự động mở trình duyệt tới http://localhost:4200/.

1. Mô tả các cấu trúc trong ứng dụng

Text

Description automatically generated with medium confidence

e2e

Thư mục này dùng để chứa các tập tin dành cho mục đích testing

## node\_modules

Chứa các module cần thiết cho ứng dụng Angular của chúng ta. Nó là thư viện của những cái trong package.json. Trong package.js là tên thư viện, còn trong node\_modules là thư viện của nó. Khi chạy lệnh npm install thì trình của mình sẽ đọc file package.json, sau đó nó lên 1 trang gọi là npm, tìm đúng tên các file trong package.json, tải về và bỏ vào node\_modules. Khi làm việc với Github thì node\_modules không up lên Github mà nó chỉ up file package.json lên thôi. Đó là lí do tại sao mà khi mình clone 1 dự án mới về, mình phải chạy lệnh npm install để lấy tất cả thư viện trong package.json cài đặt vô máy mình

## src

Đây là thư mục chính sẽ chứa toàn bộ source code của mình

Graphical user interface, text, application

Description automatically generated

### app

Đây là thư mục sẽ chứa toàn bộ code chính của mình. Nó chứa các file component, nó định nghĩa logic dữ liệu và logic ứng dụng.

app.component.html

Nơi chúng ta viết các files html. Là tầng view mà người dùng có thể thấy được.

app.component.ts (component class)

Là file xử lý các nghiệp vụ nó giống như Controller bên Spring Web.

app.component.scss

Chúng ta định nghĩa các scss mà component sẽ dùng.

app.component.ts

File này dùng cho việc testing (kiểm thử).

app.module.ts

File dùng để cấu hình cho module app.

app-routing.module.ts

File này dùng để điều hướng.

#### component

*Chứa các thành phần của Web (header, navbar, sidebar, ..)*

#### core

*Chứa các cài đặt chung cho các project của mình.*

#### layouts

*Chứa những layout chính của mình (layout admin, layout client,..)*

#### modals

*Pop mockup, dialog (bấm lên hiện lên từng ô ô)*

#### models

*Gồm những interface*

#### routes

*Những cái trang của mình (home, profile,…*

#### services

*Gọi API*

#### utils

*Chứa những hàm mình sử dụng nhiều*

### assets

Thư mục này sẽ chứa các tài nguyên như file ảnh, icons, locales (file dịch), SCSS (chứa các file SCSS config của mình).

### environments

Chứa các biến môi trường của mình (link của API, …)

### favicon.ico

Icon của ứng dụng.

### index.html

file chạy đầu tiên, Trang chủ của ứng dụng.

### main.ts

config typescript,…

### polyfill.ts

Dùng để định nghĩa các chuẩn để ứng dụng của chúng ta có thể chạy được trên mọi trình duyệt.

### style.scss

Định nghĩa style SCSS cho ứng dụng Angular, file scss lớn nhất ở ngoài cùng, import các scss

### test.ts

Code để chạy test.

## .browserslistrc

## .editorconfig

Chứa các cấu hình liên quan đến phần Editor để chỉnh sửa source code như: indent\_size, max\_line\_length,…

## .gitignore

Chứa thông tin những tập tin hoặc thư mục sẽ bị ignore không được commit lên Git Repository.

## angular.json

Đây là file config Angular của dự án mình. Đây là file xương sống của cả dự án mình.

### architect

Cấu trúc của dự án.

### build

Trước khi chạy dự án nó sẽ chạy phần build này.

## karma.conf.js

Tập tin cấu hình cho Karma, liên quan nhiều đến phần testing.

## package-lock.json

Dùng để lock version cho các Node.js module dependencies.

## package.json

File để quản lí gói

### dependencies:

Những thư viện mình cần để lập trình nên project này.

#### animations

Hiệu ứng tương tác.

#### common

#### compiler

Trình biên dịch của nó.

#### core

#### forms

Thư viện xử lí forms rất mạnh của Angular, nó mạnh nhất trong tất cả thư viện form, không có thư viện form nào qua được Angular.

#### localize

Liên quan đến ngôn ngữ.

#### platform-brower, platform-brower-dynamic

Tương tác với các nền tảng (di động, desktop,…)

#### router

Điều hướng.

#### ng-bootstrap:

thư viện UI của Angular

#### ngrx/

Liên quan đến store (team sẽ training sau).

#### translate:

Phần dịch đa ngôn ngữ.

#### bootstrap:

Thư viện bootstrap.

#### font-awsome

Về icon.

#### moment

Thư viện xử lí thời gian (ngày tháng,…).

Còn lại là các thư viện đi kèm

### devDependencies:

Nó là một cái hỗ trợ chứ không phải công cụ chính.

## README.md

Tập tin này thường được sử dụng để cho các hệ thống Git hiển thị thông tin về Git Repository của chúng ta. (thường là hướng dẫn cài đặt)

## tsconfig.app.json

## tsconfig.json

Tập tin định nghĩa việc compile cho TypeScript.

## tsconfig.spec.json

# Khai báo component trong Angular

Để tạo Component nhanh chóng thì dùng decorator có tên là @Component với cú pháp như sau:

Graphical user interface, text

Description automatically generated with medium confidence

Một decorator là một chức năng (hàm) để thêm các siêu dữ liệu (metadata), qua đó thêm thuộc tính, phương thức cho một lớp đã được định nghĩa trước. Các decorator của Angular có tiền tố là @ như: @Component, @NgModule, @Directive, @Injectable ...

## 3.1 selector

Định nghĩa một CSS selector mà Component sẽ chèn nội dung vào HTML. Nếu đặt tên như trên, thì sẽ chèn nội dung Component nếu gặp thẻ <your-selector></your-selector>, nếu muốn chỉ định như là một lớp CSS thì đặt tên ví dụ .your-selector thì nó có thể tìm đến phần tử <div class="your-selector"></div> để chèn nội dung Component

## 3.2 templateUrl

Chỉ ra file HTML làm template cho Component, nội dung File HTML này như là view của Component, nó chứa các phần tử HTML để hiện thị, nhập xuất dữ liệu ...

Bạn có thể không cần dùng đến templateUrl mà nội dung HTML nhập thẳng vào thuộc tính template

## 3.3 template

Thay vì nạp template từ file ngoài, thì toàn bộ nội dung template có thể nhập vào thuộc tính template, ví dụ:

Graphical user interface

Description automatically generated with medium confidence

Ngoài thuộc tính selector, template, templateUrl nếu muốn chỉ ra file css thì dùng thuộc tính styleUrls, style

Component được nạp vào một Module và chèn vào các View bằng Selector nó định nghĩa. Trong phần Ứng dụng Angular đầu tiên, thì ứng dụng đó có một Component là AppComponent, nó được nạp vào module và khởi chạy do thuộc boostrap của Module này (xem file src/app/app.module.ts), riêng về Module sẽ nói ở phần Module trong Angular, ở đây ta chỉ cần biết Component được một Module nạp và khởi chạy, ta xem thành phần của component AppComponent

AppComponent được khai báo tại src/app/app.component.ts

Graphical user interface, application

Description automatically generated

Component trên có selector đặt là app-root nghĩa là nội dung của Component sẽ chèn vào tại nơi khai báo thẻ <app-root></app-root>, bạn mở file index.html sẽ thấy chỗ có thẻ này!

templateUrl là app.component.html của là nội dùng file này sẽ là template (view template) của Component, nếu không thích để template ra một file thì không dùng templateUrl, mà nội dung HTMl đưa thẳng vào thuộc tính template, ví dụ:

Text

Description automatically generated

# Một số cấu trúc trong Template

## *ngIf*

Kiểm tra điều kiện \*ngIf="", nếu đúng tạo phần tử, nếu sai thì không tạo phần tử. Thay đổi ví dụ trên:

<li \*ngIf="user.name">Họ tên: {{user.name}}</li>

*ngFor*

Duyệt dữ liệu và tạo phần tử

<li \*ngFor="let item of items">

{{ item }}

</li>

Ví dụ trên, items là một dữ liệu tập hợp, mảng trong Component, duyệt qua từng phần tử của nó và tạo ra phần tử li tương ứng

*ngSwitch*

Dẽ nhiều nhánh để tạo phần tử

<div [ngSwitch]="dayofweek">

<p \*ngSwitchCase="'0'">Chủ Nhật</p>

<p \*ngSwitchCase="'1'">Thứ Hai</p>

<p \*ngSwitchDefault>Không biết</p>

</div>

Kiểm tra xem thuộc tính dayofweek khai báo trong Component, bằng bao nhiêu thì tạo phần tử p tương ứng

*ngOnInit: hàm khởi tạo chạy đầu tiên của life cycle Angular*

*contructor: nhúng những thưu viện service được sử dụng vào*

# Tạo nhanh Component từ dòng lệnh

Giờ sẽ tạo ra một Component nữa, đặt tên là ShowInfo, để nhanh chóng gõ lệnh ng sau:

 (hoặc ng g c <Tên Project>)

Nó sẽ tạo ra một Component đặt toàn bộ trong thư mục /src/app/show-info: selector là app-show-info, template là show-info.component.html

## *Thêm Component này và module AppModule*

Graphical user interface, text, application

Description automatically generated

Lúc này ở file template app.component.html chèn thêm thẻ <app-show-info></app-show-info>, thì đó là vị trí mà nội dung Component trên hiện thị, biên dịch và chạy thử để kiểm tra.

# Cách để giao tiếp giữa các component với nhau trong angular 2+

## 6.1 View child thông qua [@Input](https://viblo.asia/u/Input)

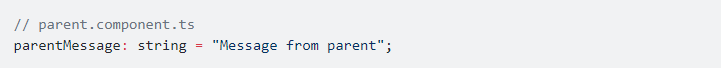
Trong ba cách thì sử dụng [@Input](https://viblo.asia/u/Input) là cách đơn giản nhất về cách khai báo cũng như sử dụng. Để để chia sẻ biến hoặc giá trị nào đó từ component cha với component con bên trong, chúng ta có thể sử dụng [@Input](https://viblo.asia/u/Input):

Ở children component: Khởi tạo:

Graphical user interface, text, application, email

Description automatically generatedThông qua biến [@Input](https://viblo.asia/u/Input) childMessage, parent component có thể truyền attribute vào children component.

Ở parent component chỉ việc khai báo:

Ở template.html:

Vậy là ParentComponent đã có thể truyền attribute tới ChildrenComponent thông qua [@Input](https://viblo.asia/u/Input).

## 6.2 EventEmitter

Cách thứ 2 là thông qua biến [@Output](https://viblo.asia/u/Output) sử dụng EventEmitter:

Event emitter được thiết kế để báo cho component cha khi component con có sự thay đổi.

Thông qua biến [@Output](https://viblo.asia/u/Output), EventEmitter sẽ bắn một value nào đó ra ngoài, và component cha sẽ bắt được value này.

* Ở component con:

 Graphical user interface, text, application, email

Description automatically generated

* **Ở component cha:**

## Graphical user interface, text, application, email Description automatically generated6.3 Data service BehaviourSubject

Data service sử dụng BehaviourSubject (rxjs).

Rxjs hỗ trợ observes (consuming interface) và observables (push interface) (có bài viết đọc khá dễ hiểu, các bạn có thể tham khảo thêm [ở đây](https://viblo.asia/p/rxjs-nhap-mon-oaKYMN1zR83E)).

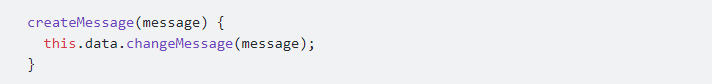
Ở ví dụ sau mình sử dụng cả 2 interface trên của BehaviourSubject, thông qua hàm asObservable()

File data.service.ts:

Graphical user interface, text, application, email

Description automatically generatedBây giờ để chia sẻ data giữa các component, chúng ta cần import DataService:

Gọi hàm changeMessage để push data:

Và subcribe vào currentMessage để get data:



1. **Vòng đời (circle life) của component trong Angular**

Vòng đời của một component hay một directive trong Angular tính từ lúc nó được tạo ra, nó bị thay đổi và bị phá huỷ. Hiểu được vòng đời của component ta có thể viết code can thiệp trong quá trình component hay direct được tạo ra, được cập nhật và phá huỷ.

Khi ứng dụng Angular được start lên thì đầu tiên nó sẽ tạo và render component cha (hay còn gọi là root component) sau đó nó sẽ tạo và render các component con. Khi mỗi component được load lên, component sẽ kiểm tra xem có data binding vào nó không, dữ liệu có thay đổi không và cập nhật lại chúng. Khi component bị phá huỷ thì chúng sẽ bị remove (xoá) khỏi giao diện web.

Angular cung cấp cho chúng ta một số phương thức về vòng đời của một component. Dựa vào đó chúng ta có thể can thiệp vào quá trình tạo ra component, cập nhật giá trị và phá huỷ của component. Sau đây là thứ tự từ trên xuống dưới về các method sẽ được sử dụng trong vòng đời của một component.

* ngOnChanges
* ngOnInit
* ngDoCheck
* ngAfterContentInit
* ngAfterContentChecked
* ngAfterViewInit
* ngAfterViewChecked
* ngOnDestroy

## ****7.1 ngOnChanges****

Phương thức ngOnChanges được gọi khi component phát hiện có giá trị được binding vào component bằng phương pháp Input properties. Để nhận biết được giá trị binding thì ngOnChange quản lý đối tượng SimpleChange. Chúng ta sử dụng @Input trong bài truyền giá trị từ cha xuống con là một ví dụ.

Ví dụ sau ta có component cha truyền giá trị message xuống cho component con (child-component) thông qua properties sau đó component con sẽ dùng @Input để nhận giá trị và binding vào component.

|  |  |
| --- | --- |
| 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 | import { Component} from '@angular/core';  import { Customer } from './customer';    @Component({  selector: 'app-root',  template: '  <h1>!</h1>  <p> Message : <input type='text' [(ngModel)]='message'> </p>  <p> Code : <input type='text' [(ngModel)]='code'></p>  <p> Name : <input type='text' [(ngModel)]='name'></p>  <p><button (click)="updateCustomer()">Update </button>  <child-component [message]=message [customer]=customer></child-component>  ' ,    styleUrls: ['./app.component.css']  })  export class AppComponent {  title = 'ngOnChanges';  message = '';  customer: Customer = new Customer();  name= '';  code= 0;    updateCustomer() {  this.customer= new Customer();  this.customer.name = this.name;  this.customer.code = this.code;  }    } |

Chúng ta có component con với nội dung như sau:

|  |  |
| --- | --- |
| 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 | import { Component, Input, OnInit, OnChanges, SimpleChanges, SimpleChange,ChangeDetectionStrategy } from '@angular/core';  import { Customer } from './customer';    @Component({  selector: 'child-component',  template: '<h2>Child Component</h2>  <p>Message </p>  <p>Customer Name </p>  <p>Customer Code <p>  <ul><li \*ngFor="let log of changelog;"> </li></ul> '  })  export class ChildComponent implements OnChanges, OnInit {  @Input() message: string;  @Input() customer: Customer;  changelog: string[] = [];    ngOnInit() {  console.log('OnInit');  }    ngOnChanges(changes: SimpleChanges) {  console.log('OnChanges');  console.log(JSON.stringify(changes));    // tslint:disable-next-line:forin  for (const propName in changes) {  const change = changes[propName];  const to = JSON.stringify(change.currentValue);  const from = JSON.stringify(change.previousValue);  const changeLog = `${propName}: changed from ${from} to ${to} `;  this.changelog.push(changeLog);  }  }  } |

Đầu tiên chúng ta import thư viện từ angular core như sau:

|  |  |
| --- | --- |
| 1 | import { Component, Input, OnInit, OnChanges, SimpleChanges, SimpleChange } from '@angular/core'; |

Tiếp đến ta cài đặt các vòng đời của Angular

|  |  |
| --- | --- |
| 1  2  3  4 | export class ChildComponent implements OnChanges, OnInit {  @Input() message: string;  @Input() customer: Customer;  changelog: string[] = []; |

Trong hàm ngOnChanges ta lấy tất cả các sự thay đổi giá trị của component thông qua đối tượng SimpleChange.

|  |  |
| --- | --- |
| 1  2  3 | ngOnChanges(changes: SimpleChanges) {  console.log('OnChanges');  console.log(JSON.stringify(changes)); |

Xử lý nghiệp vụ trong hàm ngOnChange cho phù hợp với yêu cầu của chúng ta.

## ****7.2 ngOninit****

Phương thức NgOninit được gọi, khi component được tạo lần đầu tiên. Chúng ta được chạy sau khi hàm constructor và hàm ngOnchange được thực hiện.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { Component, OnInit } from '@angular/core';    @Component({  selector: 'child-component',  template: '  <h2>Child Component</h2>  ' ,  styleUrls: ['./app.component.css']  })  export class ChildComponent implements OnInit {    constructor() {  console.log('ChildComponent:Constructor');  }    ngOnInit() {  console.log('ChildComponent:OnInit');  }  } |

Đầu tiên chúng ta import thư viện OnInit từ angular core.

|  |  |
| --- | --- |
| 1 | import { Component, OnDestroy, OnInit } from '@angular/core'; |

Tiếp đến chúng ta khai báo cài đặt hàm OnInit.

|  |  |
| --- | --- |
| 1 | export class ChildComponent implements OnInit { |

Cuối cùng chúng ta viết code trong phương thức ngOnInit để can thiệp vào lúc component được tạo ra.

|  |  |
| --- | --- |
| 1  2  3 | ngOnInit() {  console.log('ChildComponent:OnInit');  } |

## ****7.3 ngOnDoCheck****

Phương thức ngOnDoCheck được gọi mỗi khi nó phát hiện ra có sự thay đổi dữ liệu ở component.

|  |  |
| --- | --- |
| 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  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | import { Component, Input, OnChanges, OnInit, SimpleChanges, SimpleChange, DoCheck } from '@angular/core';  import { Customer } from './customer';    @Component({  selector: 'child-component',  template: `<h2>Child Component</h2>  <p>Message </p>  <p>Customer Name </p>  <p>Customer Code </p>  <p>Do Check count </p>  <ul><li \*ngFor="let log of changelog;"> </li></ul> `  })  export class ChildComponent implements OnChanges, DoCheck, OnInit {  @Input() message: string;  @Input() customer: Customer;  changelog: string[] = [];  oldCustomer: Customer= new Customer();  DocheckCount = 0;    ngOnInit() {  console.log('OnInit');  this.oldCustomer = Object.assign({}, this.customer);  }    ngDoCheck() {  console.log('Docheck');  this.DocheckCount++;  if (this.oldCustomer.name !== this.customer.name || this.oldCustomer.code !== this.customer.code ) {  const to = JSON.stringify(this.customer);  const from = JSON.stringify(this.oldCustomer);  const changeLog = `DoCheck customer: changed from ${from} to ${to} `;  this.changelog.push(changeLog);    this.oldCustomer = Object.assign({}, this.customer);  }  }    ngOnChanges(changes: SimpleChanges) {  console.log('OnChanges');  console.log(JSON.stringify(changes));    // tslint:disable-next-line:forin  for (const propName in changes) {  const change = changes[propName];  const to = JSON.stringify(change.currentValue);  const from = JSON.stringify(change.previousValue);  const changeLog = `${propName}: changed from ${from} to ${to} `;  this.changelog.push(changeLog);  }  } |

Đầu tiên chúng ta import DoCheck từ thư viện angular core.

|  |  |
| --- | --- |
| 1 | import { Component, Input, OnChanges, OnInit, SimpleChanges, SimpleChange, DoCheck } from '@angular/core'; |

Chúng ta implement DoCheck interface.

|  |  |
| --- | --- |
| 1 | export class ChildComponent implements OnChanges, DoCheck, OnInit { |

Cuối cùng chúng ta viết code trong hàm ngDoCheck để kiểm tra sự thay đổi và thêm các dòng code mình muốn**.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | ngDoCheck() {  console.log('Docheck');  this.DocheckCount++;  if (this.oldCustomer.name !== this.customer.name || this.oldCustomer.code !== this.customer.code ) {  const to = JSON.stringify(this.customer);  const from = JSON.stringify(this.oldCustomer);  const changeLog = `DoCheck customer: changed from ${from} to ${to} `;  this.changelog.push(changeLog);    this.oldCustomer = Object.assign({}, this.customer);  } |

## ****7.4 ngAfterContentInit****

Phương thức này được gọi sau khi component được khởi tạo thành công.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Component({selector: 'my-cmp', template: `...`})  class MyComponent implements AfterContentInit {  ngAfterContentInit() {  // ...  }  } |

## ****7.5 ngAfterContentChecked****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Phương thức này được gọi sau khi Component Content được kiểm tra bởi Angular's Change module.  @Component({selector: 'my-cmp', template: `...`})  class MyComponent implements AfterContentChecked {  ngAfterContentChecked() {  // ...  }  } |

## ****7.6 ngAfterViewInit****

Nó tượng tự nhưng ngAfterContentInit but nó được gọi khi component và các component con của nó được khởi tạo thành công. Chỉ được gọi 1 lần sau khi ngAfterContentChecked.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Component({selector: 'my-cmp', template: `...`})  class MyComponent implements AfterViewInit {  ngAfterViewInit() {  // ...  }  } |

## ****7.7 ngAfterViewChecked****

Phương thức này được gọi sau khi các view component cha và các view của component con được kiểm tra xong.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Component({selector: 'my-cmp', template: `...`})  class MyComponent implements AfterViewChecked {  ngAfterViewChecked() {  // ...  }  } |

## ****7.8 ngOnDestroy****

Phương thức ngOnDestroy được gọi trước khi component hoặc directive bị phá huỷ bởi Angular.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import { Component, OnDestroy } from '@angular/core';    @Component({  selector: 'child-component',  template: '  <h2>Child Component</h2>  ',  styleUrls: ['./app.component.css']  })  export class ChildComponent implements OnDestroy {    constructor() {  console.log('ChildComponent:Constructor');  }        ngOnDestroy() {  console.log('ChildComponent:OnDestroy');  }    } |

Đầu tiên chúng ta import OnDestroy tử angular core.

|  |  |
| --- | --- |
| 1 | import { Component, OnDestroy } from '@angular/core'; |

Tiếp đến implement OnDestroy Interface

|  |  |
| --- | --- |
| 1 | export class ChildComponent implements OnDestroy |

Cuối cùng là viết code trong hàm ngOnDestroy

|  |  |
| --- | --- |
| 1  2  3 | ngOnDestroy() {  console.log('ChildComponent:OnDestroy');  } |

# TÀI LIỆU THAM KHẢO

[1] Angular, <https://angular.io/docs>, truy cập lần cuối ngày 20/10/2021.

[2] Trung Nguyen (20/07/2021). Không gian làm việc và cấu trúc file dự án, <https://comdy.vn/angular/khong-gian-lam-viec-va-cau-truc-file-du-an/>, truy cập lần cuối ngày 20/10/2021.

[3] Nguyễn Mạnh Hùng (23/1/2018). Giới thiệu về Angular CLI, <https://itprotraining.vn/vi/tips-cong-nghe/angularcli>, truy cập lần cuối ngày 20/10/2021.

[4] Cơ chế hoạt động trong Angular, <https://levunguyen.com/laptrinhweb/2021/06/03/co-che-hoat-dong-trong-Angular/>, truy cập lần cuối ngày 20/10/2021.

# MỤC LỤC

[1. Cài đặt 1](#_Toc85639134)

[1.1 Cài đặt ứng dụng Angular Cli 1](#_Toc85639135)

[1.2 Tạo Project bằng Angular Cli 1](#_Toc85639136)

[1.3 Chạy ứng dụng 1](#_Toc85639137)

[2. Mô tả các cấu trúc trong ứng dụng 1](#_Toc85639138)

[node\_modules 1](#_Toc85639139)

[src 1](#_Toc85639140)

[.browserslistrc 4](#_Toc85639141)

[.editorconfig 4](#_Toc85639142)

[.gitignore 4](#_Toc85639143)

[angular.json 4](#_Toc85639144)

[karma.conf.js 4](#_Toc85639145)

[package-lock.json 4](#_Toc85639146)

[package.json 4](#_Toc85639147)

[README.md 6](#_Toc85639148)

[tsconfig.app.json 6](#_Toc85639149)

[tsconfig.json 6](#_Toc85639150)

[tsconfig.spec.json 6](#_Toc85639151)

[TÀI LIỆU THAM KHẢO 9](#_Toc85639152)