**TÌM HIỂU VỀ ANGULAR**

├── e2e

│ ├── src

│ │ ├── app.e2e-spec.ts

│ │ ├── app.po.ts

│ ├── protractor.conf.js

│ ├── tsconfig.e2e.json

├── node\_modules

├── src

│ ├── app

│ │ ├── app-routing.module.ts

│ │ ├── app.component.css

│ │ ├── app.component.html

│ │ ├── app.component.spec.ts

│ │ ├── app.component.ts

│ │ ├── app.module.ts

│ ├── assets

│ │ ├── .gitkeep

│ ├── environments

│ │ ├── environment.prod.ts

│ │ ├── environment.ts

│ ├── favicon.ico

│ ├── index.html

│ ├── main.ts

│ ├── polyfills.ts

│ ├── styles.css

│ ├── test.ts

├── .editorconfig

├── .gitignore

├── angular.json

│── browserslist

├── karma.conf.js

├── package-lock.json

├── package.json

├── README.md

├── tsconfig.app.json

├── tsconfig.json

├── tsconfig.spec.json

└── tslint.json

Ở thư mục root của Angular project, các bạn sẽ thấy nhiều thư mục và tập tin, mục đích của những tập tin thư mục này mình có thể liệt kê như trong bảng dưới đây:

|  |  |
| --- | --- |
| **Tập tin / Thư mục** | **Mục đích** |
| e2e | Thư mục này dùng để chứa các tập tin dành cho mục đích testing |
| node\_modules | Chứa các Node.js module cần thiết cho ứng dụng Angular |
| src | Đây là thư mục sẽ chứa toàn bộ source code của ứng dụng Angular |
| .editorconfig | Chứa các cấu hình liên quan đến phần Editor để chỉnh sửa source code như: indent\_size, max\_line\_length,… |
| .gitignore | Đây là tập tin metadata của Git, chứa thông tin những tập tin hoặc thư mục sẽ bị ignore không được commit lên Git Repository. |
| angular-cli.json | Đây là tập tin chứa cấu hình cho Angular CLI, giúp chúng ta có thể build ứng dụng Angular. |
| 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 | Tập tin cấu hình cho Node.js module dependencies |
| protractor.conf.js | Tập tin cấu hình cho Protractor, liên quan nhiều đến phần testing |
| 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. |
| tslint.json | Tập tin cấu hình để kiểm tra lỗi cho các tập tin .ts (TypeScript) trong Angular project |
| tsconfig.json | Tập tin định nghĩa việc compile cho TypeScript. |
| browserslistrc | Cài đặt các trình duyệt, nền tảng project hỗ trợ |

Trong các tập tin, thư mục được liệt kê ở trên thì thư mục src là thư mục quan trọng, là nơi để chúng ta có thể thêm code, modify code để phát triển ứng dụng Angular của mình.

Mục đích của từng thư mục, tập tin trong thư mục src này mình có thể liệt kê như sau:

|  |  |
| --- | --- |
| **Tập tin / Thư mục** | **Mục đích** |
| app | Đây là thư mục sẽ chứa toàn bộ code của ứng dụng Angular |
| assets | Thư mục này sẽ chứa các file ảnh, CSS, custom JavaScript của ứng dụng Angular |
| environments | Chúng ta có thể viết ứng dụng chạy trên nhiều môi trường khác nhau, đây chính là thư mục giúp chúng ta làm định nghĩa các tập tin cấu hình cho những môi trường khác nhau đó |
| favicon.ico | Icon của ứng dụng Angular |
| index.html | Trang chủ của ứng dụng Angular |
| main.ts | Chứa code bootstrapping cho ứng dụng Angular |
| polyfill.ts | Dùng để định nghĩa các chuẩn để ứng dụng của chúng ta có thể chạy được trên mọi trình duyệt |
| style.css | Định nghĩa style CSS cho ứng dụng Angular |
| test.ts | Code để chạy test |
| tsconfig.json | Tập tin định nghĩa việc compile cho TypeScript |

Trong một component có các tập tin sau:

|  |  |
| --- | --- |
| **Tập tin / Thư mục** | **Mục đích** |
| app.component.css | file css của component |
| app.component.html | file html của component |
| app.component.spec.ts | file sử dụng cho unit test |
| app.component.ts | file xử lí logic |
| app.module.ts | file dùng để cấu hình cho module app |
| app-routing.module.ts | file này dùng để điều hướng |
| **Tập tin / Thư mục** | **Mục đích** |
| core | Chứa các cài đặt chung cho trang web |
| layout | Chứa các layout chính |
| modals | Chứa các modal, popup |
| models | Chứa các interface |
| routes | Chứa các trang (home,..) |
| service | Gọi API |
| utils | Chứa hàm được sử dụng nhiều |

**Cách tạo component sử dụng Angular CLI**

ng generate component <component-name>

Tệp app.component.ts:

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'Ứng dụng Angular đầu tiên';

}

*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 <app-root></ app-root>, 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

*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 ... Có thể không cần dùng đến templateUrl mà nội dung HTML nhập thẳng vào thuộc tính template

@Component({

  selector: 'app-root',

  template: '<h1>Hello World</h1>'

})

Trong TypeScript để nhập chuỗi nhiều dòng trong code thì đặt chuỗi trong cặp ``

@Component({

  selector: 'app-root',

  template: `

  <h1>Hello World!</h1>

  <p>This template definition spans multiple lines.</p>

  `

})

*styleUrls*: Khai báo các kiểu thành phần sử dụng cho mẫu của nó theo một trong hai cách: bằng cách tham chiếu đến tệp bên ngoài dùng *styleUrls* hoặc trực tiếp trong thành phần dùng *styles*.

@Component({

  selector: 'app-root',

  template: `

  <h1>Hello World!</h1>

  <p>This template definition spans multiple lines.</p>

  `,

  styles: ['h1 { font-weight: normal; }']

})

Thuộc tính styles nhận một mảng chuỗi chứa các khai báo quy tắc CSS.

**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*

**Tương tác Component**

* *Truyền dữ liệu từ Component cha sang Component con với Input binding @Input*

HeroChildComponent có 2 thuộc tính đầu vào.

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

import {Hero} from './hero';

@Component({

    selector: 'app-hero-child',

    template: `

    <h3>{{hero.name}} says:</h3>

    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>

    `

})

export class HeroChildComponent {

    @Input() hero!: Hero;

    @Input('master') masterName = '';

}

@Input() hero!: Hero, Component cha sẽ thiết lập bằng thuộc tính hero thông qua thuộc tính HTML cùng tên trong HTML [hero]

Input(‘master’) masterName = ‘’, thuộc tính Component con masterName được Component cha thiết lập thông qua thuộc tính HTML [master]

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

import {HEROES} from './hero';

@Component({

    selector: 'app-hero-parent',

    template: `

    <h2>{{master}} controls {{heroes.length}} heroes</h2>

    <app-hero-child \*ngFor="let hero of heroes"

    [hero]="hero"

    [master]="master">

    </app-hero-child>

    `

})

export class HeroParentComponent {

    heroes = HEROES;

    master = 'Master';

}

* *Thay đổi thuộc tính đầu vào với setter*

Set giá trị mặc định đối với giá trị đầu vào

* *Thay đổi thuộc tính đầu vào với ngOnchanges()*

Phát hiện thay đổi ở Component cha sau đó thực hiện hành động ở Component con đối với thuộc tính giá trị đầu vào bằng phường thức OnChanges()

* *Component cha lắng nghe sự kiện của Component con*

Component con hiển thị thuộc tính EventEmitter mà nó phát ra sự kiện khi có điều gì đó xảy ra. Component cha liên kêt thuộc tính sự kiện đó và phản ứng lại.

Thuộc tính EventEmitter của con là thuộc tính output @Output()

* *Component cha tương tác với con sử dụng local variable*
* *Component cha gọi @ViewChild()*
* *Component cha và con giao tiếp qua service*

**Lệnh import**

Lệnh import chỉ ra cho Angular biết tìm các function đó ở đâu và sử dụng trong module của chúng ta. Tất cả các thành phần bên ngoài giống như một thư viện của third party, các module của Angular phải được import. Một lệnh import là một phần của đặc tả ES2015. Nó tương tự như như lệnh import của Java hay lệnh Using của C#.

Bạn chỉ có thể import chỉ các member được export từ module khác.

**Class**

Class chứa logic của ứng dụng. Nó có thể chứa method và properties như là C# hoặc Java class. Class phải được định nghĩa với từ khoá export nếu bạn muốn sử dụng class từ module khác.

**Class Decorator**

Nếu không có class decorator, AppComponent chỉ là một class. Không có gì liên quan Angular. Nó là decorator chỉ ra cho Angular biết cách sử dụng class đó.

Ví dụ, @Component decorator chỉ ra cho Angular biết class là một Component. Tương tự thế, một @Directive kể cho Angular biết class đó là Directive. Angular hiện tại có các class decorator:

@Component

@Directive

@Injectable

@NgModule

@Pipe

**Xây dựng các khối của ứng dụng Angular**

Nhìn vào ứng dụng Task Application trong section trước, bạn có thể định danh 7 khối của Angular như sau:

Component

Templates

Metadata

Data Binding

Directives

Services

Dependency Injection

*Component*

Angular Component là một class nó được bổ trợ bởi @Component decorator. Component điều khiển một phần của UI.

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

title = 'GettingStarted';

}

Component có 4 phần như sau:

Import statement

Class

Template

Metadata

Lệnh import các thành phần liên quan được yêu cầu bởi Component này. Class chứa các logic của ứng dụng. Nó được bổ sung thông tin bởi @Component decorator.

*Template*

Component cần một view để hiển thị. Template định nghĩa view. Template chỉ là một tập con của HTML, nó chỉ cho Angular biết làm sao để hiển thị view. Nó là một trang HTML chuẩn sử dụng các thẻ H1, H2...Nó cũng sử dụng các lệnh của Angular như {}, []...

*Metadata*

Metadata chỉ cho Angular làm sao để xử lý class. Chúng ta thêm Metadata vào class sử dụng class decorator. Khi chúng ta thêm @Component decorator vào class nó sẽ trở thành component.

Class decorator sử dụng đối tượng cấu hình, cung cấp thông tin Angular cần để tạo component. Ví dụ @Component directive đi với cấu hình như selector, templateURL, directive...

*Data Binding*

Angular sử dụng Data Binding để lấy dữ liệu từ Component đến View. Nó được hoàn thành sử dụng cú phpas HTML đặc tả được biết đến là Template Syntax

Angular hỗ trợ 4 kiểu của Data binding:

Interpolation: Data được bind ra từ component sang view

Property Binding: Data bind ra từ component sang thuộc tính HTML trong view.

Event Binding: Các sự kiện DOM được bind ra từ view ra method của Component

Two-way binding/Model binding: Luồng dữ liệu cả hai chiều từ view sang component và ngược lại.

*Directive*

Directive giúp chúng ta thao tác với View. Một directive là một class, chúng ta tạo ra sử dụng @Directive nó chứa metadata và logic thao tác trên DOM.

View được tạo ra bởi Angular sử dụng template metadata được định nghĩa trong Component. Các template là động và được chuyển đổi sử dụng directive.

Angular hỗ trợ 2 loại directive là Structural directive sẽ thay đổi cấu trúc của view. Còn loại kia là attribute directive sẽ chỉnh sửa style của view.

*Services*

Service cung cấp dịch vụ cho các Components hoặc các service khác. Angular không có bất cứ đặc tả nào cho Service nó chỉ là các class có export method và chứa một số task.

export class MyLogger {

AddTolog(msg: any)

{

console.log(msg);

}

}

Và trong bất cứ component nào, chúng ta chỉ cần gọi nó:

log :MyLogger = new MyLogger();

constructor() {

this.log.AddTolog("Component Created");

}

Nó là các Javascript module thuần không có gì đặc biệt. Angular đã làm gì để các service này có sẵn cho component? Nó gọi là cơ chế dependecy injection.

*Dependency Injection*

Dependency Injection là một phương thức khi tự động tạo một instance của một service được inject vào một component hoặc một service khác khi nó được yêu cầu.

Denpendency injection được dùng hầu hết khi inject service vào component hoặc service khác.

Angular làm điều này sử dụng injector. Khi một component được tạo, Angular sẽ tìm Component metadata các service mà component này yêu cầu. Injector sẽ tạo ra instance của service đó và inject vào component sử dụng constructor của nó.

Nếu service đã tạo thì injector sẽ không tạo mà sử dụng lại. Service cần chỉ ra cho Angular biết nó sẵn sàng được inject vào bất cứ component nào yêu cầu nó bằng cách sử dụng @Injectable decorator.

@Injectible()

export class MyLogger {

AddTolog(msg: any)

{

console.log(msg);

}

}

Và giờ MyLogger class có thể được inject vào bất cứ component nào hoặc service nào.