Angular: framework for building client applications in html/css and javascript/TypeScript

Why we need angular: because with vanilla Js and jQuery is har to maintain and teste in a big project

structure of angular app:

* e2e : end to end test for application
* node\_modules : this is where we store all the third party libraries that application made depend on

when we compile our application parts of this third-party libraries are put in a bundle and deployed with our application

* src : actual source code of the application.
  + app : contain model and component, each application has at list on model and one component.
  + assets : here we store all images file, text files ,icons …
* environments:
  + store configuration setting of environments
* main file

starting point of the application / bootstrapping the main module

* polyfills.ts

import some file required for angular

* styles.css

the global style of our application

* test.ts

setting test of the application

webpack:

* angular CLI use webpack tool to build automatization tool.
* Get all script and stylesheet and combine them in a bundle and then minified them for optimization
* Web pack recompile the application and refresh the bundle / hot module replacement (HMR)
* All stylesheets are compiled in javascript bundel

Angular Histories :

AngularJs - Angular2 - Angular4

1. **Typescript Fundamentals**

* Any java script code is a Typescript /Type script has additional feature than JavaScript.
* Typescript: Strong Typing (Opt) , OOP, Compile-time errors, Great tooling

typescript -- transpile 🡪 JavaScript

* tsc main.ts génère un fichier java script.
* node main.js
* Modules : in type script each file is a module

**export**

export class Point {

**import**

import { Point } from './point';

1. **Angular Fundamentals**

Building Blocks of angular Apps

* **Component**: Data, Html Template, Logic
* App component is a root component
* **Modules**: is container of group of related components every angular app has at list one module which we call angular Module

Ex: courses model, Messaging model, instructor model, admin model…

To use a component there is tree steps to follow:

* 1. **Create** a component
  2. **Register** it in a module
  3. Add an element in a **HTML markup**
* **String Interpolation**

<h2>{{ getTitle() }}</h2>'

* **Directive:**

 <li \*ngFor="let course of courses"> {{course}} </li>

* **Service:**

ng d s [nom de service]

* **Property Binding**
* Bind a property of the DOM to a filed in a component.
* One-way binding from component to the dom.
* Property binding work only for DOM object NOT HTML element.
* To target a DOM attribute with property binding we use the following syntax

<h1 [textContent]=”title”></h1>

* **Attribute Binding**
* To target html attribute with property binding we use the following syntax

<td [attr.colspan]=”colspan”></td>

* **Bootstrap**

ˆ3.3.7: major.minor.patch

**Download all dependency**: npm install

* **Class Binding**

[class.active]="\_isActive"

* **Style Binding**

<button class="btn btnprimary" [style.background]="\_isActive ? 'blue' : 'white'">btn</button>

* **Event binding**

onSave($event){

    $event.stopPropagation();

    console.log("Button was clicked",$event.clientX);

}

* **Event filtering**

   <input (keyup.enter)="enter()"/>

enter($event){

console.log($event.target.value);

}

* **Template variables**

 <input #email (keyup.enter)="getEmail(email.value)"/>

getEmail(text){

    console.log(text);

}

* **Two-way binding**

     <input [(ngModel)]="\_email" (keyup.enter)="printEmail()"/>

\_email="mohamedamjoud1@gmail.com";

courses;

printEmail(){

console.log(this.\_email);

}

* **Pipes**

{{ course.title | uppercase}} <br/>

{{ course.students | number }} <br/>

{{ course.rating | number: '2.1-2' }} <br/>

{{ course.price | currency: 'AUD':true:'3.2-2' }} <br/>

{{ course.releaseDate | date }} <br/>

1. **Building Re-usable Components**

In order to make a component reusable you want add input and output property.

input property: to pass input or state to a component

output property: to raise event from this custom component

the combination between input and output property make up the public API a component

* **Input**

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

@Input() isFavorite :  boolean = false;

* **Aliasing Input Properties (give to a property a nickName)**

@Input('is-Favorite') isFavorite :  boolean = false;

* **Output**

*component*

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

@Output() change = new EventEmitter();

 onClick(){

    this.isFavorite = !this.isFavorite;

    this.change.emit();

  }

*Call the component*

<course [isFavorite]="post.isFavorite" (change)="onFavoriteChange()"></course>

* **Output - Passing event data**

*Component*

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

@Output() change = new EventEmitter();

onClick(){

  this.isFavorite = !this.isFavorite;

  this.change.emit({ newValue : this.isFavorite });

}

export interface FavoriteChangedEventArgs{

  newValue : boolean;

}

*Call the component in html page*

<course [isFavorite]="post.isFavorite" (change)="onFavoriteChange($event)">

</course>

*Definition of* ***onFavoriteChange***

onFavoriteChange(eventArgs : FavoriteChangedEventArgs){

  console.log("Favorite changed" + eventArgs.newValue);

}

* **Styles**

We define style of component in three ways:

By using styleUrls,By using styles Or by using style markup in html file

@Component({

  selector: 'course',

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

  styleUrls: ['./course.component.css'],

  styles:[

  `

  .fa-star{

    color : red;

    font-size : 50px;

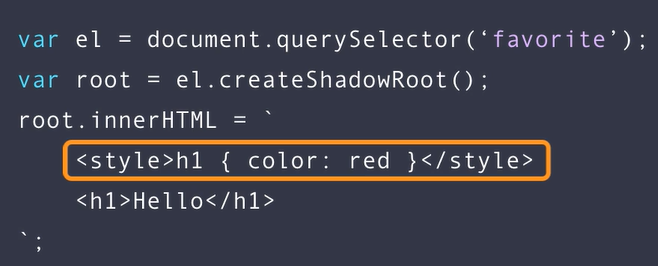
  }

  `]

})

* **Shadow DOM**

Allows us to apply scoped styles to elements without bleeding out to the outer world.



*Apply shadow behavior*

 encapsulation: ViewEncapsulation.Emulated

 encapsulation: ViewEncapsulation.Non

*to avoid*

 encapsulation: ViewEncapsulation.Native

* **ng-content**

Allow to consumer of the component to provide custom content

Definition of component

<div class="card">

    <div class="card-header">

        <ng-content select=".heading"></ng-content>

    </div>

    <div class="card-body">

        <ng-content select=".body"></ng-content>

    </div>

</div>

Call component and custom content

<bootstrap-panel>

    <div class="heading">Heading</div>

    <div class="body">

        <h1>Body</h1>

        <p>Some content here ...</p>

    </div>

</bootstrap-panel>

* **ng-container**

we use it to costume a component when we don’t need to add extra markup

(div, span …)

    <ng-container class="heading">Heading</ng-container>

1. **Directives**

There is tow type of directives structural and attribute.

Structural: Modify the structure of the DOM

Attribute: Modify the attributes of DOM elements.

* **ng-if**