Angular

João Gonçalves

Interactive Developer / Trainer / Consultant

Applied Mathematics Degree

Master Multimedia Communication

http://joaogoncalves.net

edu@joaogoncalves.net

You ?

ES6

ECMAScript

ECMAScript is a standard for client-side scripting languages

ECMAScript

first edition of the ECMAScript specification was released in 1997

ES5 (standard) released in 2009

Sixth edition was finalized in 2015

ES6 or ES2015

ES7 in 2016, ES8 in 2017, …, ES2024 (jun2024)

ES6

Scope, let, const, template literal, multiline string, optional parameters, arrow functions, rest, spread, generator functions, destructuring, classes, ...

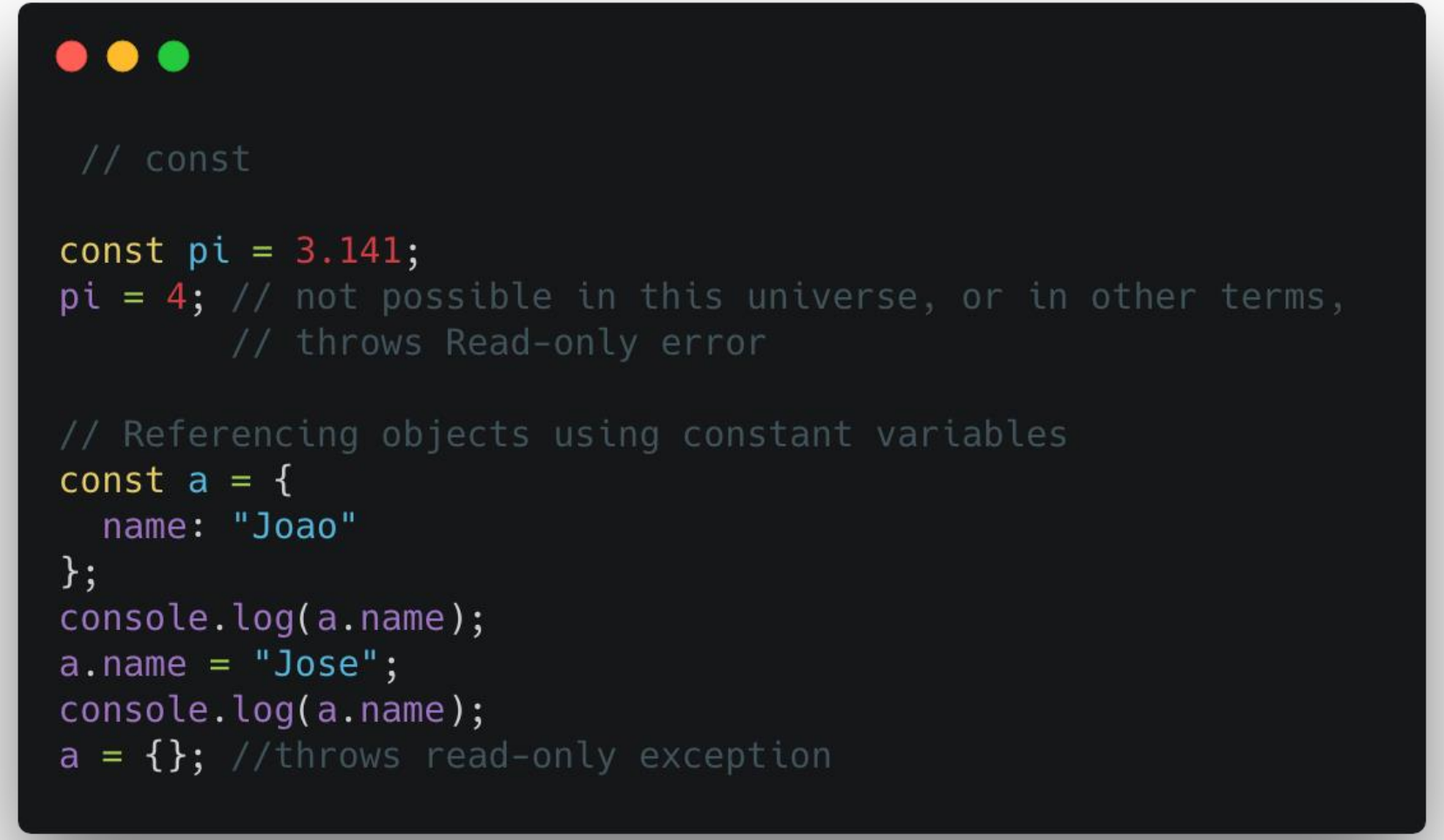
Let / const (blocked scoped)

```
// scope

let a = 12; // accessible everywhere
 function myFunction() {
   console.log(a); // alerts 12
   let b = 13;
   if(true) {
     let c = 14; // this is ONLY accessible inside if block
     alert(b); // alerts 13
   }
    alert(c); // alerts undefined
 }
 myFunction();
 alert(b); // alerts undefined
```

Redeclaring variables

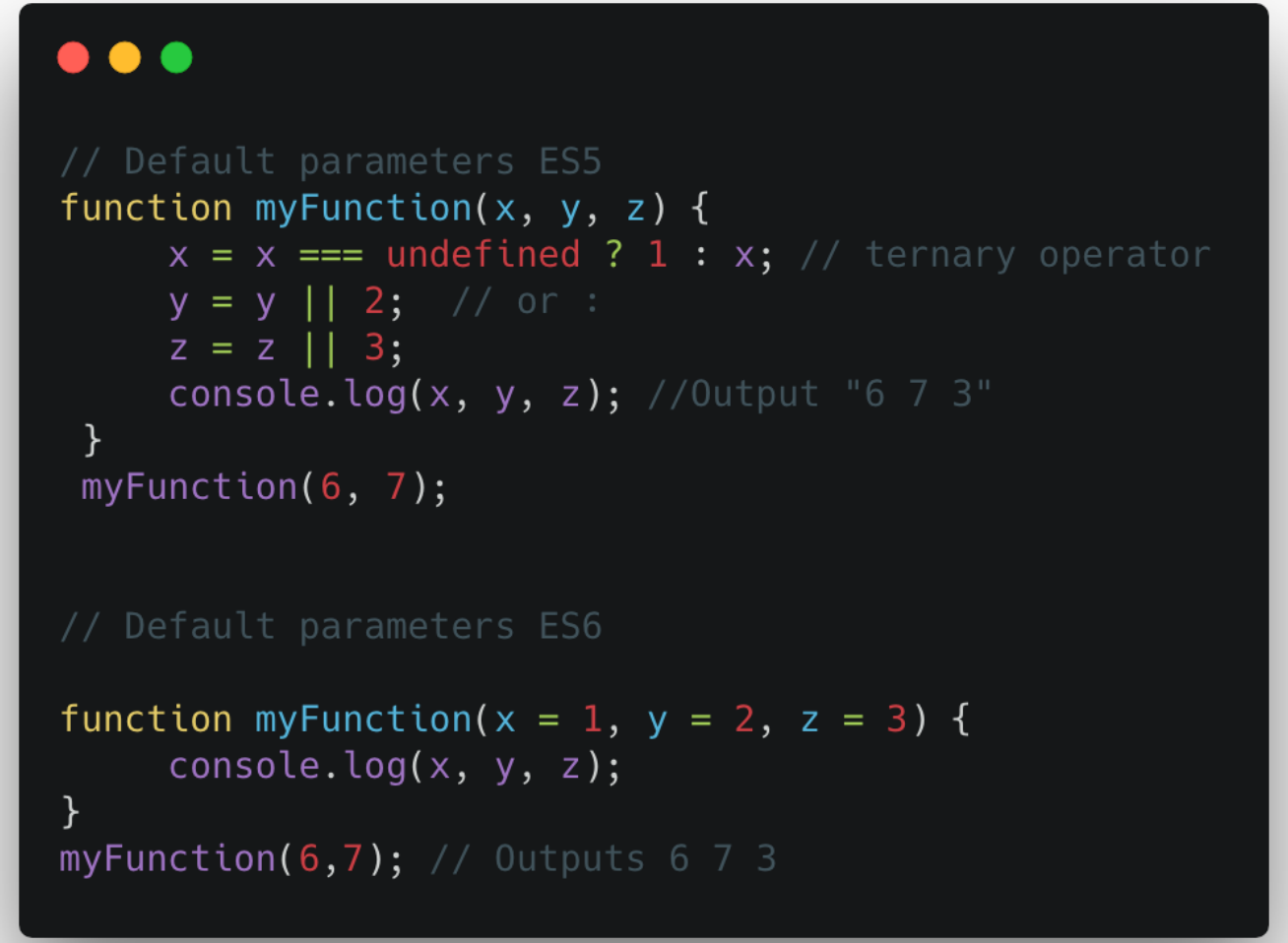
```
// redeclaring variables

let a = 0;
let a = 1; // SyntaxError:  Identifier 'a' has already been declared
function myFunction() {
 let b = 2;
 let b = 3; // SyntaxError:  Identifier 'b' has already been declared
 if(true) {
    let c = 4;
    let c = 5; // SyntaxError:  Identifier 'c' has already been declared
 }
}
myFunction();
```

Const (blocked scoped)



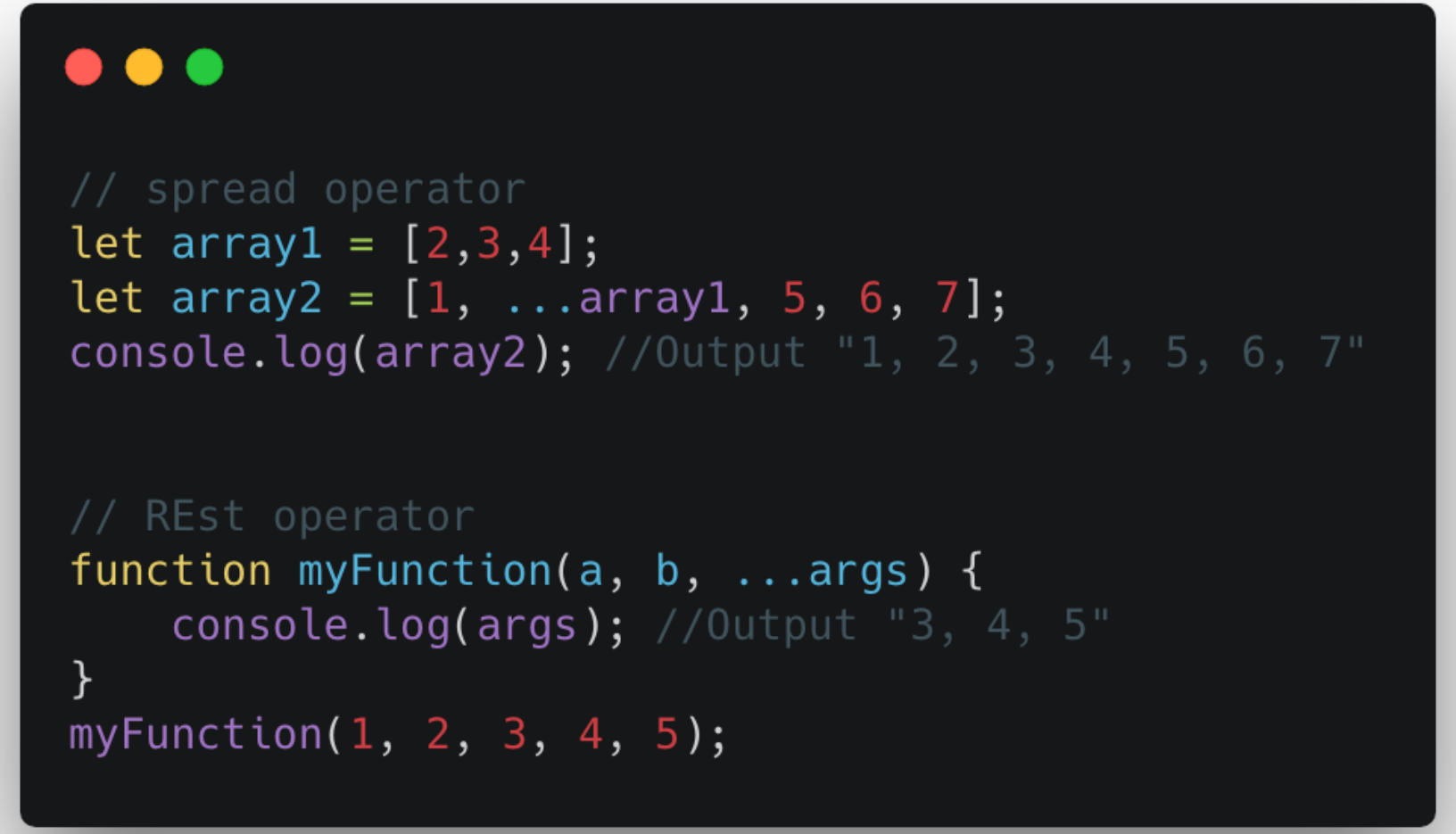
```
 // const

const pi = 3.141;
pi = 4; // not possible in this universe, or in other terms,
        // throws Read-only error

// Referencing objects using constant variables
const a = {
  name: "Joao"
};
console.log(a.name);
a.name = "Jose";
console.log(a.name);
a = {}; //throws read-only exception
```

Default parameters



```
// Default parameters ES5
function myFunction(x, y, z) {
     x = x === undefined ? 1 : x; // ternary operator
     y = y || 2;  // or :
     z = z || 3;
     console.log(x, y, z); //Output "6 7 3"
 }
 myFunction(6, 7);


// Default parameters ES6

function myFunction(x = 1, y = 2, z = 3) {
     console.log(x, y, z);
}
myFunction(6,7); // Outputs 6 7 3
```

Spread & rest operators

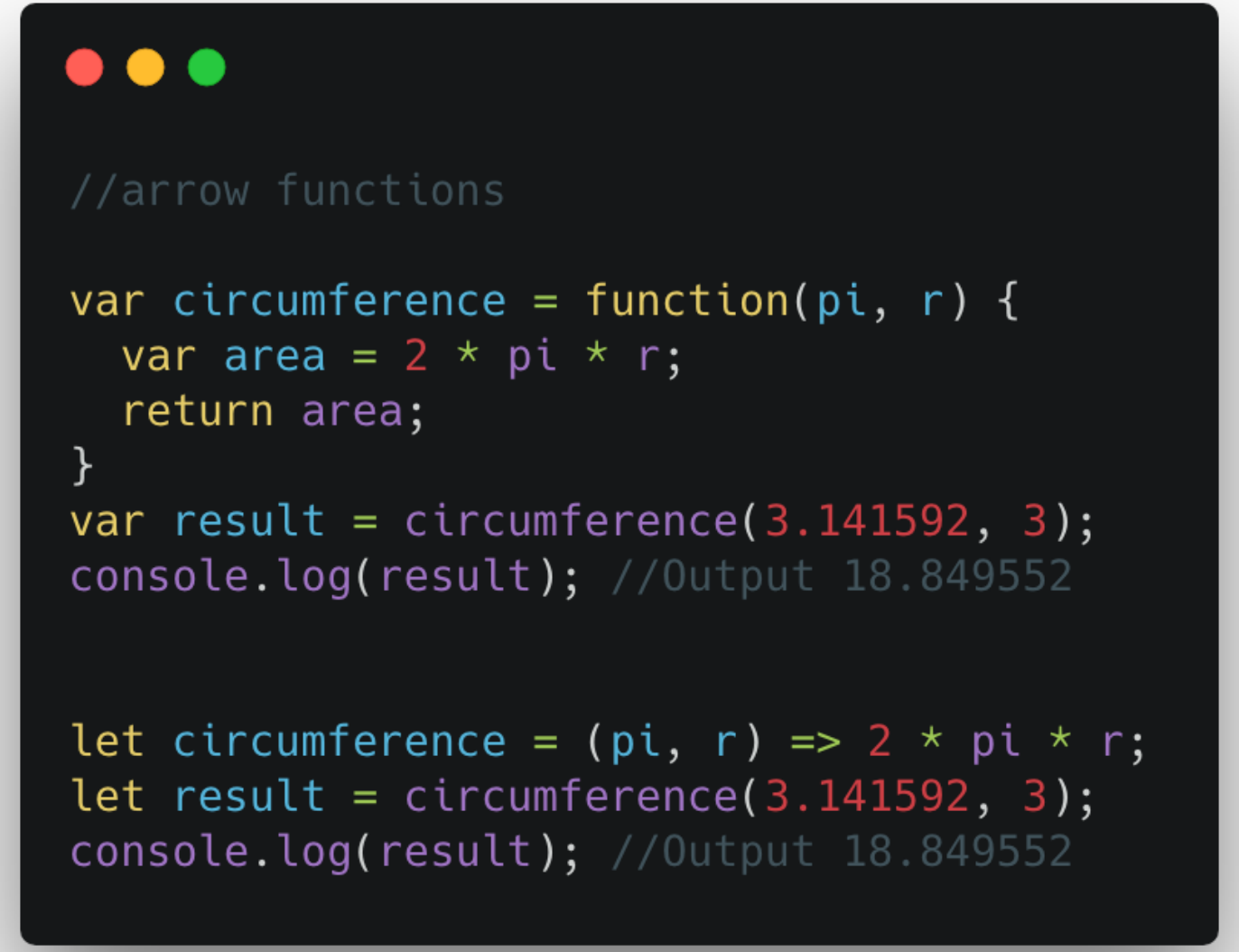


```
// spread operator
let array1 = [2,3,4];
let array2 = [1, ...array1, 5, 6, 7];
console.log(array2); //Output "1, 2, 3, 4, 5, 6, 7"


// REst operator
function myFunction(a, b, ...args) {
    console.log(args); //Output "3, 4, 5"
}
myFunction(1, 2, 3, 4, 5);
```

Destructuring Arrays & Objects

```
// destructuring Arrays

let myArray = [1, 2, 3];
let a, b, c;
[a, b, c] = myArray;  //array destructuring assignment syntax

// or
let [a, b, c] = [1, 2, 3];

//with rest operator
 let [a, ...b] = [1, 2, 3, 4, 5, 6];
   console.log(a);   //  1
   console.log(Array.isArray(b));   //  true
   console.log(b);  // 2,3,4,5,6

// destructuring Objects
let object = {"name" : "John", "age" : 23};
let name, age;
({name, age} = object); //object destructuring assignment syntax
```

Arrow functions



```
//arrow functions

var circumference = function(pi, r) {
  var area = 2 * pi * r;
  return area;
}
var result = circumference(3.141592, 3);
console.log(result); //Output 18.849552


let circumference = (pi, r) => 2 * pi * r;
let result = circumference(3.141592, 3);
console.log(result); //Output 18.849552
```

Template Literals, Multiline Strings

```
//template literals

const a = 20;
const b = 10;
const c = "JavaScript";
const str = `My age is ${a+b} and I love ${c}`;
console.log(str);
```

ES6 – livros online

https://eloquentjavascript.net/

http://exploringjs.com/

Preparação Ambiente de trabalho

Node.js: https://nodejs.org

Git: https://git-scm.com

Typescript: "*npm install -g typescript*"

VSCode: https://code.visualstudio.com

Angular DevTools: https://angular.dev/tools/devtools

History…

Angular is a **TypeScript-based** Javascript framework.

Developed and maintained by Google

“Superheroic JavaScript **MVW** Framework”

History…

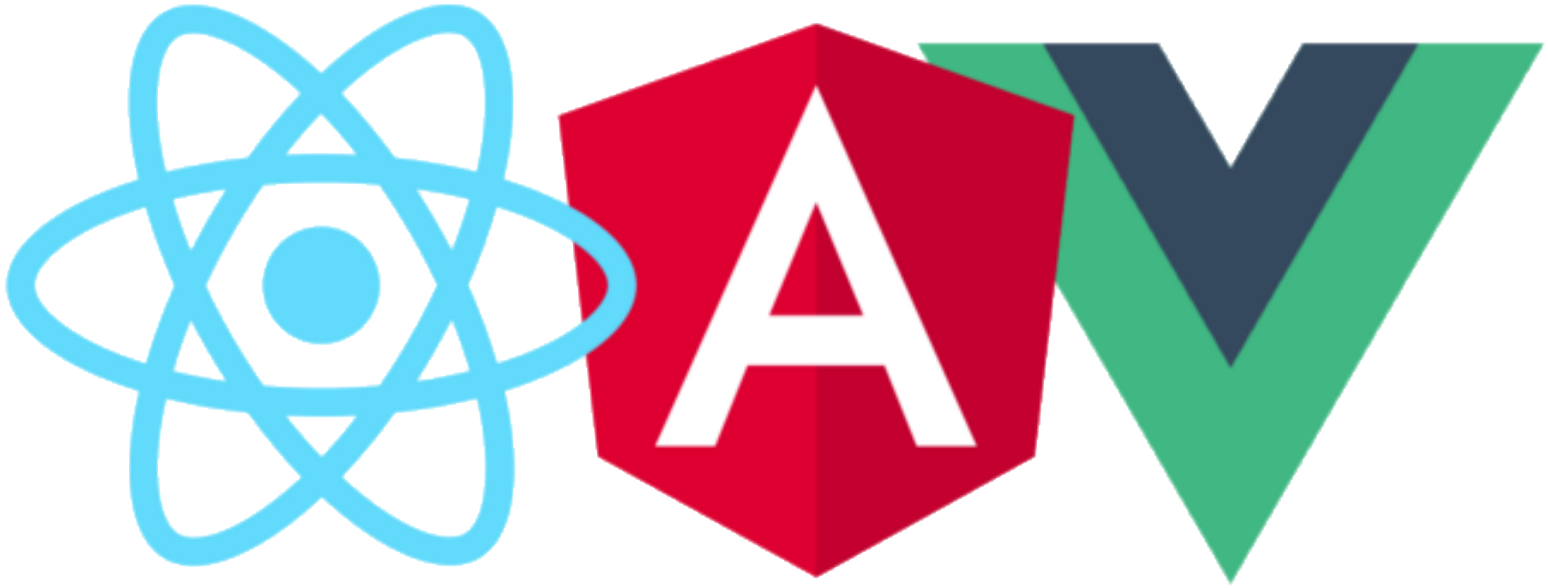
Angular (2016)

(also “Angular 2+”, “Angular 2” or “ng2”)

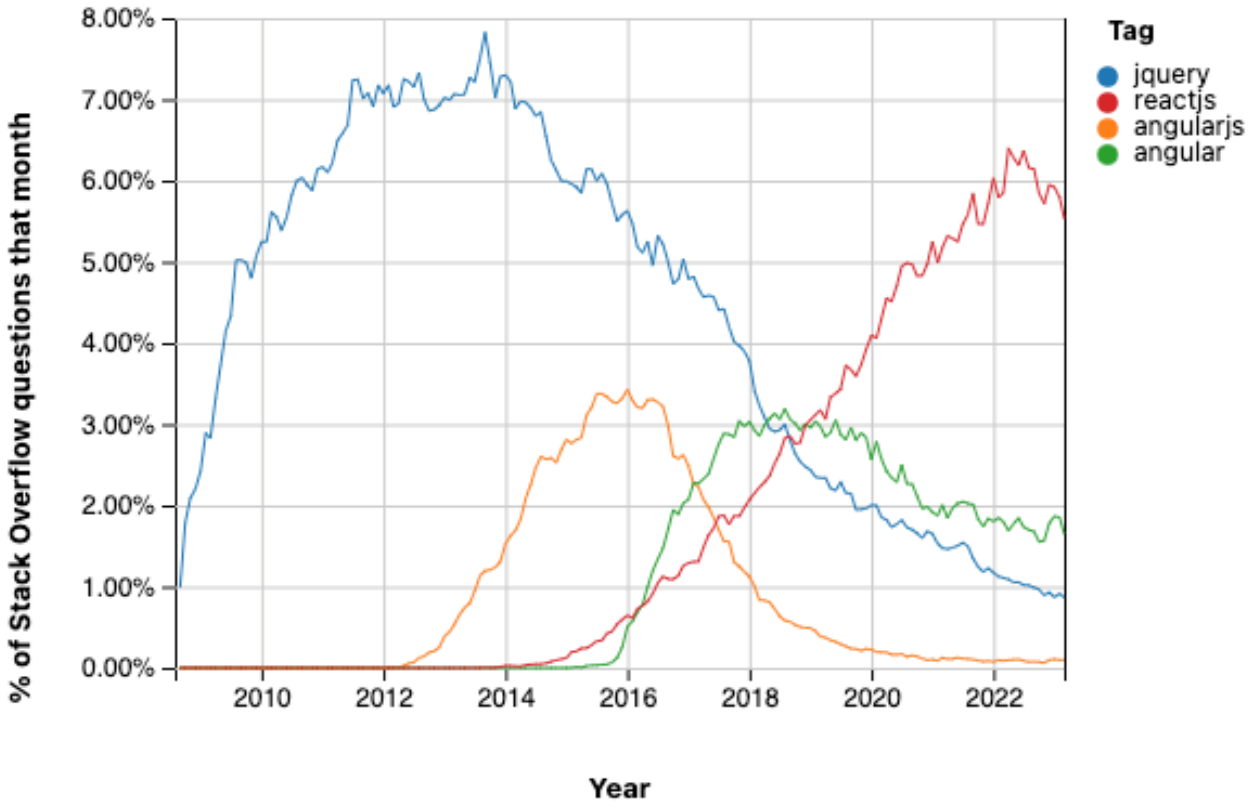
is the rewritten,

mostly incompatible successor to **AngularJS(2010)**

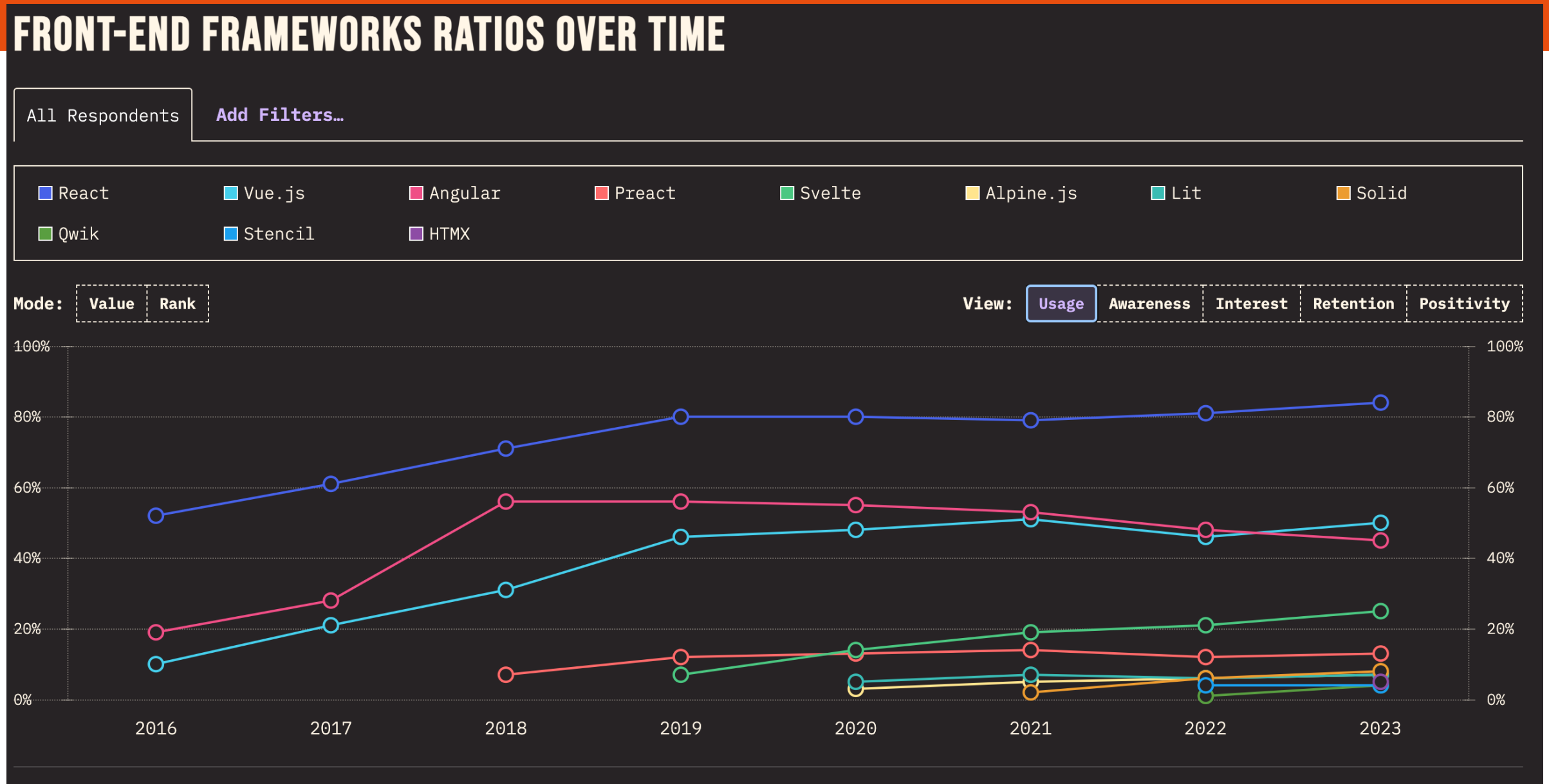
History…



History…


8.00%
7.00%
6.00%
5.00%
4.00%
3.00%
2.00%
1.00%
0.00%
% of Stack Overflow questions that month
2010
2012
2014
2016
2018
2020
2022
Year
Tag
jquery
reactjs
angularjs
angular

State Of JS ...


FRONT-END FRAMEWORKS RATIOS OVER TIME
All Respondents
Add Filters…
React
Vue.js
Angular
Preact
Svelte
Alpine.js
Lit
Solid
Qwik
Stencil
HTMX
Mode:
Value
Rank
View:
Usage
Awareness
Interest
Retention
Positivity
100%
80%
60%
40%
20%
0%
2016
2017
2018
2019
2020
2021
2022
2023

Estatisticas..

Ver the stateofjs

https://stateofjs.com/

Why Angular

With **React** or **Vue**, you'll also need to select other products for **routing, dependency injection, forms, bundling and deploying** …

The **Angular** framework is a platform that includes **all you need for developing and deploying** a Web app

Generate a new single-page web app in seconds using **Angular CLI**

web app that consists of a **set of components** that can communicate with each other in a loosely-coupled

Why Angular... Features..

Arrange the client-side navigation using the powerful **router**

Inject and easily replace **services**

Arrange **state management** via injectable **singleton services (DI)**

Why Angular... Features..

Cleanly separate the **UI and business logic**

Modularize your app so only the core functionality is loaded on app startup
(Lazy Loading)

modern-looking UI using the **Angular Material library**

Why Angular... Features..

Implement **reactive programming** where your app components do not pull the data that may not be ready yet, but simply **subscribe to data source and get notifications** where the data is available

Angular CLI

is a **tool** for **managing** Angular projects

is a **code generator** that simplifies the process of **project creation** as well **generating new components, services**, and **routes** in existing app

Modern Angular

Angular, as one of the most popular front-end development frameworks

The framework has seen years of **improvements in performance**, **user experience**, and **new features**, like the introduction of the **Ivy rendering** engine which led to the reduction of bundle sizes and runtime improvements

Now, the community can **focus on more than just improving visible parts** of the framework, the parts that directly **impact user experience**

Modern Angular

attention can now be **directed toward the developer experience**. This includes **better scalability and composability** among other aspects

the Angular team has delivered several **important updates in recent versions**

which have become **important breakthroughs**, putting Angular on a path of almost **revolutionary changes, and improvements** are going to be the **topics that we will discuss** in this training

Modern Angular

This training assumes knowledge of Angular, TypeScript, and HTML

Technology	Level	Details
Typescript	Basic	Knowledge of what TypeScript is, how to declare types of variables, functions, and objects, and knowledge about generic types
Angular	intermediate	familiarity with the building blocks of an Angular application (components, directives, etc), and knowledge about Angular's built-in packages like Http, Routing
RXJS	Basic	basic knowledge of RxJS is necessary, mainly knowledge about Observables, operators, and subscriptions
HTML	Basic	The most entry-level knowledge of HTML tags and attributes is enough
CSS	Basic	Knowledge of CSS selectors is enough

Modern Angular – How Angular Was (Core Features)

Object Oriented Programming

First of all, most Angular building blocks like **components**, **pipes**, **directives**, **guards**, and many more, have been historically authored with OOP

represented as **classes** could be confusing for some developers, especially those coming from other popular front-end frameworks like React

Modern Angular – How Angular Was (Core Features)

Dependency Injection (DI)

Until recently, **DI** has been completely coupled with classes and OOP (@Injectable)

We will see, that with the **addition of the new inject function** this constraint (using exclusively classes) has evaporated, opening a **new era of composability and reusability**

Modern Angular – How Angular Was (Core Features)

Module-based architecture

Before v14, all Angular applications were **built around NgModule**, an Angular-specific concept

will see how Angular **now** allows building applications **without NgModule**
a new practice now known as **“standalone”**

Modern Angular – How Angular Was (Core Features)

RxJS

the reactive extensions library for JavaScript, most likely plays a huge part in **sharing the state between different parts of an Angular application**

we shall see **new features** that dramatically increase the interoperability between Angular applications and RxJS

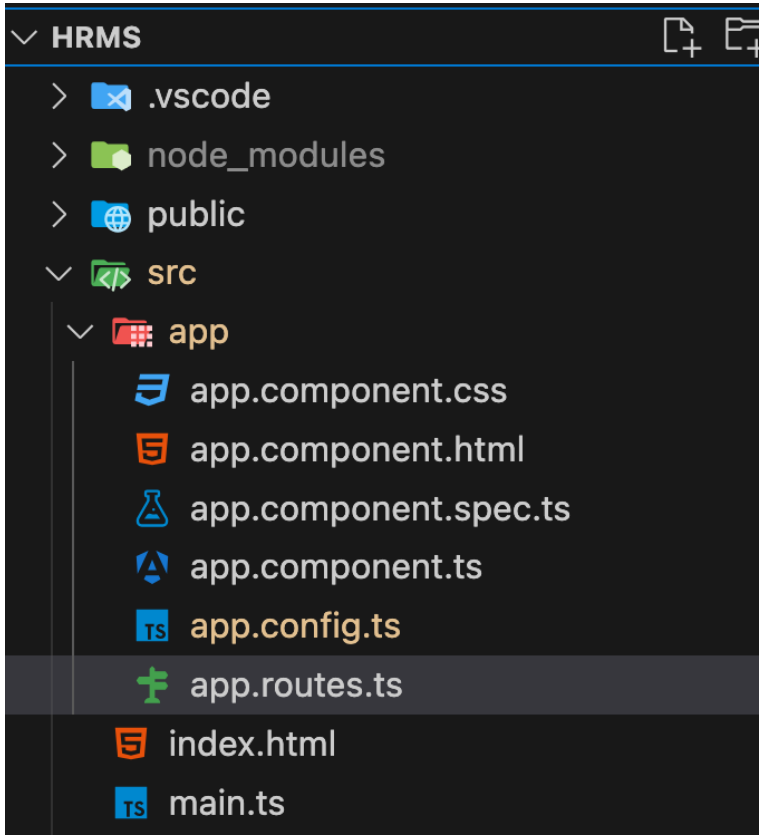
Modern Angular – How Angular Was (Core Features)

Change detection

the mechanism by which **Angular propagates the changes** in a component's data to the UI, is a **pretty complex** and somewhat **sub-optimal algorithm (zone.js)**

learn how to create solutions without that, and dive deep into the **change detection mechanism**

Modern Angular – Structure Changes

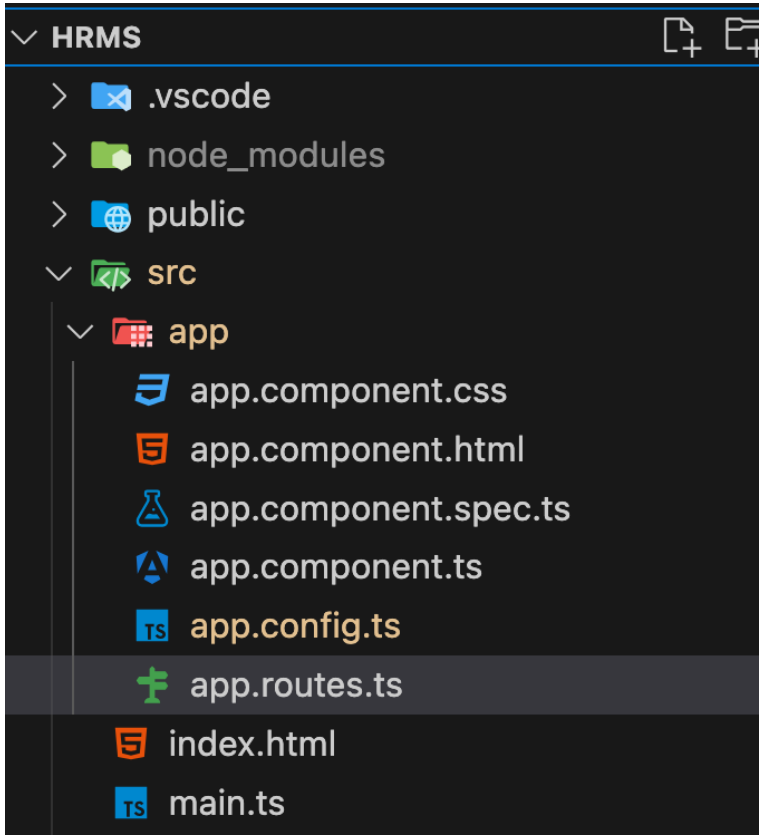


No "environments" folder: from Angular v15, environment files are not generated by default and can be added via a separate command

No explicit **"polyfills.ts"** file projects

angular.json file, we will notice it is far **shorter** than we used to have in older

Modern Angular – Structure Changes

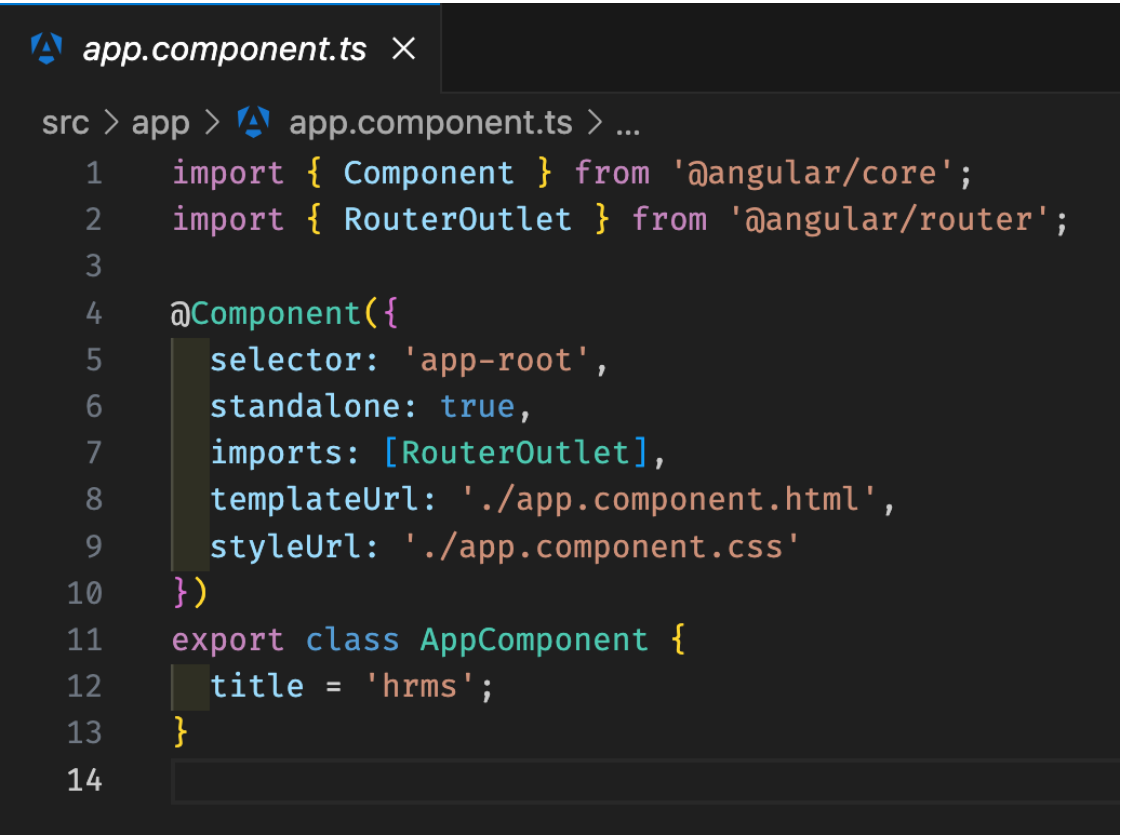


No app.module.ts file: The application is fully standalone and does not utilize modules for its architecture

app.routes.ts file instead of app.routing.module.ts: this is again because we chose standalone

app.config.ts file: this file will contain global configurations for our app, like providers, routing initialization

Modern Angular – Stand Alone Component



standalone: true marks this component as standalone and not belonging to any NgModule

The imports array is used to import its dependencies

Modern Angular - Routes

app.routes.ts

src > app > app.routes.ts > ...

```ts
import { Routes } from '@angular/router';

export const routes: Routes = [];

```

app.config.ts M

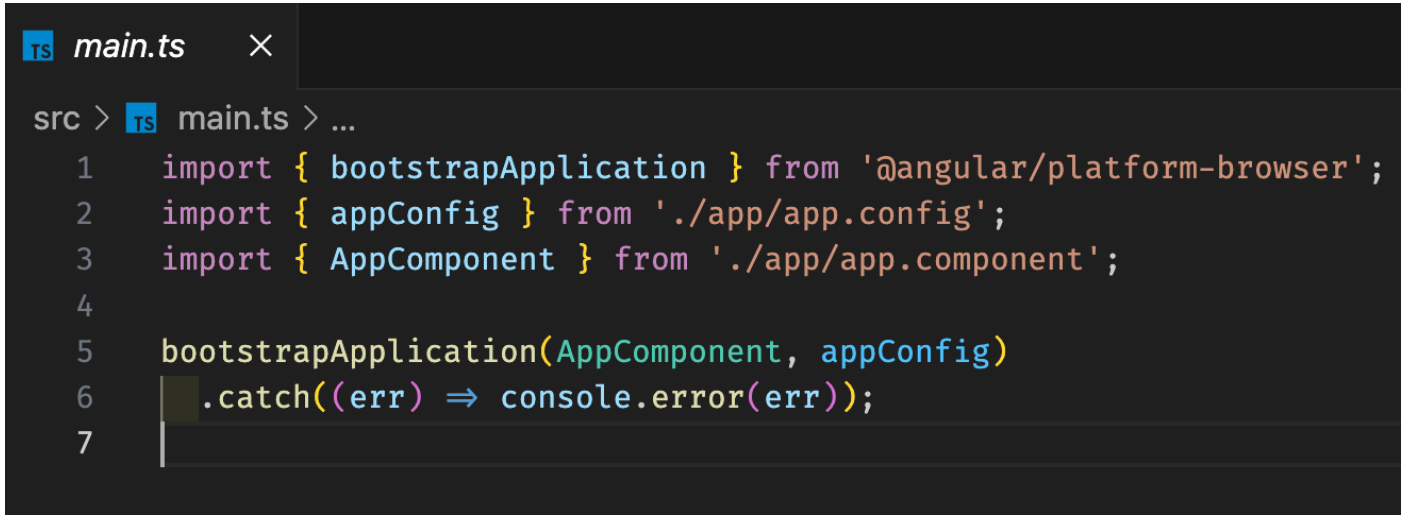
src > app > app.config.ts > ...

```ts
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes)
  ]
};

```

does not use the RouterModule to register routes.

routes are only defined here and registered in the **app.config.ts** file

Modern Angular - Bootstrapp



our application is being initialized and **bootstrapped** in the **main.ts** file

In **modern Angular apps**, we do not need an NgModule

this **new function** can directly create our application using one root component and the application configuration.

Modern Angular – New Features

Standalone building blocks

Starting from **v14** (stable in v15), having NgModule is no longer a requirement

Angular building blocks can **now be "standalone",** meaning they do not require an associated NgModule to be used in an app

Modern Angular – New Features

The inject function

Until v14, it was only possible to inject dependencies in classes that were marked with one of Angular's decorators

With this **new function**, we can overcome this limitation, and build a **composable, reusable function** that can be **easily shared between components**

allowing for never before heard **composability**

Modern Angular – New Features

Type-safe Reactive Forms

from v14, new Typed Reactive Forms have been introduced
that are also now marked as ”stable”

Directive composition API

new **hostDirectives property** has been added to the component/directive metadata object, essentially allowing us to **build directives from other directives**

Modern Angular – New Features

Better compatibility with RxJS

rxjs-interop, has been added to Angular, which will help the developers integrate RxJS code seamlessly into Angular apps

allowing switching from Signals to Observables and vice-versa

Modern Angular – New Features

Signals

Probably **the most impactful addition to Angular ever**, signals are a new reactive primitive that are called **to solve common problems we face with RxJS**

and transform and significantly improve the change detection mechanism

Modern Angular – New Features

New template syntax

Starting from v17, a new template syntax is available that is projected to replace ngIf, ngSwitch, ngFor directives

better readable templates and compiler optimizations

Modern Angular – New Features

Deferred loading of parts of a template

Another addition to the new template syntax allows deferred loading of a part of a template, either based on a condition or an event

Modern Angular – New Features

New tools for unit testing

The addition of a new unit testing framework, support for new API-s (like the above-mentioned inject function)

Modern Angular – New Features

Server-side rendering hydration

the server side has long been one of the weakest points of Angular

full hydration, greatly improving the **performance of SSR apps** allowing reuse of the existing application state and DOM

Modern Angular – New Features

Various granular improvements to performance

Different small tools that improve the loading of the page and its different parts, like the loading of images

A Standalone Future

Modern Angular – standalone future

Using Angular components/directives/pipes without NgModules

Structuring applications without NgModules

Routing and lazy-loading of standalone components

Modern Angular – standalone future

Why abandon NgModules?

NgModule-s still exist and are supported, deprecation is not even yet discussed

Standalone building blocks interop with NgModule-s just fine

This is more about making NgModule optional rather them getting rid of them completely (for now)

The core team itself seems to favor standalone which makes future deprecation more likely

Modern Angular – standalone future

Why abandon NgModules?

Hard to learn, hard to explain

layer of confusion, as there is already a concept of a module in TypeScript and JavaScript

Indirectness and boilerplate

Modern Angular – standalone future

Developing apps without NgModules

It is possible to build applications completely standalone

The standalone approach is backward compatible, so NgModule-s and standalone components can (and often do) coexist in the same codebase

Lots of third-party tools and libraries still have not migrated away from NgModule

The structure of an Angular application

The structure of an Angular application

Creating our first component

Interacting with the template

Component inter-communication

Encapsulating CSS styling

Deciding on a change detection strategy

Introducing the component lifecycle

The structure of an Angular application

Let’s create our first application:

“ng new my-app”

ng serve -o

The structure of an Angular application

vscode: Includes VSCode configuration files

node_modules: Includes installed npm packages that are needed to develop and run the Angular application

public: Contains static assets such as fonts, images, and icons

src: Contains the source files of the application

.editorconfig: Defines coding styles for the default editor

.gitignore: Specifies the files and folders that Git should not track

angular.json: The main configuration file of the Angular CLI workspace

The structure of an Angular application

package.json and package-lock.json: Provide definitions of npm packages, along with their exact versions, which are needed to develop, test, and run the Angular application

README.md: A README file that is automatically generated from the Angular CLI

tsconfig.app.json: A TypeScript configuration that is specific to the Angular application

tsconfig.json: A TypeScript configuration that is specific to the Angular CLI workspace

tsconfig.spec.json: A TypeScript configuration that is specific to unit tests of the Angular application

The structure of an Angular application

When we develop an Angular application, we'll likely interact with the ***src*** folder:

app: All the Angular-related files of the application. You interact with this folder most of the time during development.

index.html: The main HTML page of the Angular application.

main.ts: The main entry point of the Angular application.

styles.css: CSS styles that apply globally to the Angular application. The extension of this file depends on the stylesheet format you choose when creating the application.

The structure of an Angular application

app folder contains the actual source code we write for our application:

app.component.css: Contains CSS styles specific to the sample page. The extension of this file depends on the stylesheet format you choose when creating the application.

app.component.html: Contains the HTML content of the sample page.

app.component.spec.ts: Contains unit tests for the sample page.

app.component.ts: Defines the presentational logic of the sample page.

app.config.ts: Defines the configuration of the Angular application.

app.routes.ts: Defines the routing configuration of the Angular application.

The structure of an Angular application

Vscode …

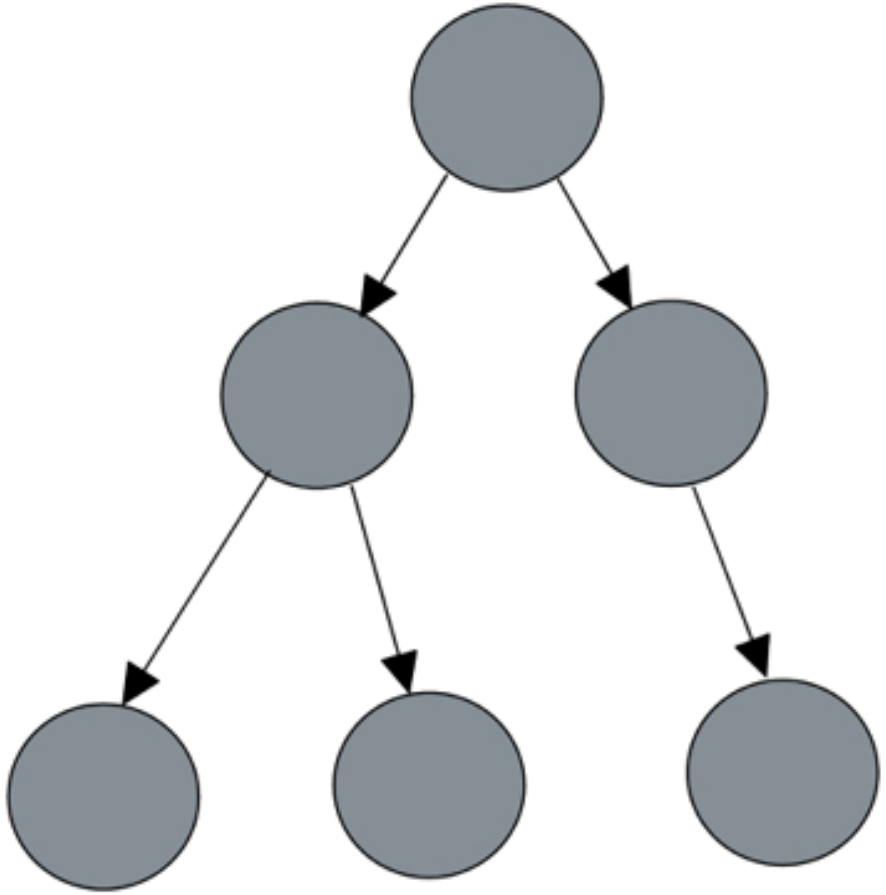
Structuring User Interfaces with Components

Components are the basic **building blocks** of an Angular application

They are responsible for the **presentational logic** of an Angular application

organized in a **hierarchical tree** of components that can interact with each other

can **communicate and interact with one or more components** in the component tree



Structuring User Interfaces with Components

a typical **Angular application** contains **at least a main component**
that consists of multiple files (or not)

Code ..

ng generate component product-list

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'World';
}
```

Structuring User Interfaces with Components

Controlling data representation:

- Displaying data conditionally - @if()
- Iterating through data - @for()
- Switching through templates

(CODE…)

Component inter-communication

Angular components expose a public API that
allows them to communicate with other components

This API encompasses input properties

also exposes output properties we can bind event listeners

"Code: Let's create a "product Details" component…"

Encapsulating CSS styling

We can define CSS styling within our components to better encapsulate our code and make it more reusable

CSS styles apply to the elements contained in the component, but they do not spread beyond their boundaries

Angular embeds style sheets in the <head> element of a web page

Encapsulating CSS styling

We can set up different levels of view encapsulation

Emulated: Entails an emulation of native scoping in shadow DOM by sandboxing the CSS rules under a specific selector that points to a component.

Native: Uses the native shadow DOM encapsulation mechanism of the renderer that works only on browsers that support shadow DOM.

None: Template or style encapsulation is not provided. The styles are injected as they were added into the <head> element of the document

Code ..

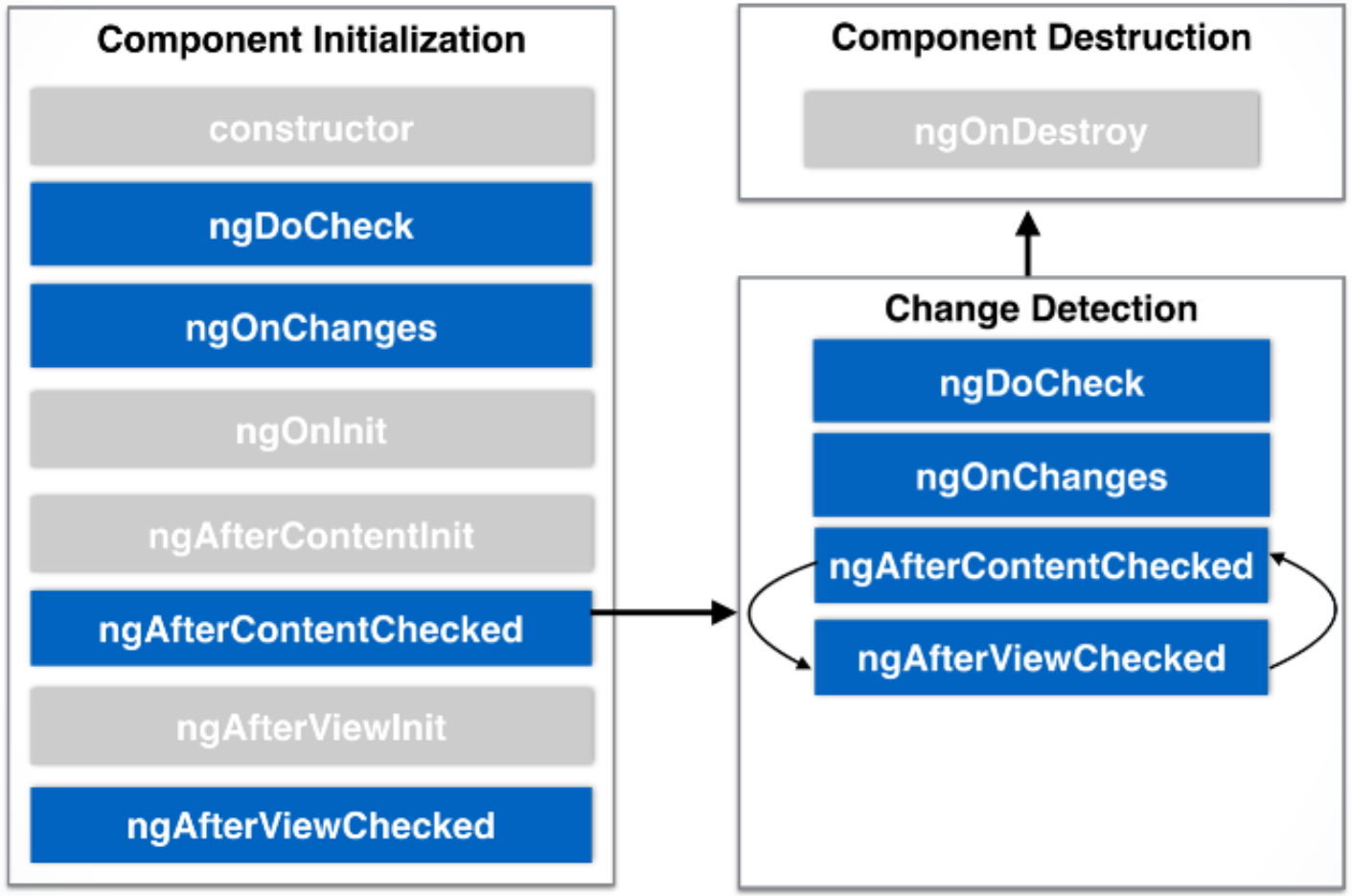
COMPONENT LIFECYCLE

COMPONENT LIFECYCLE

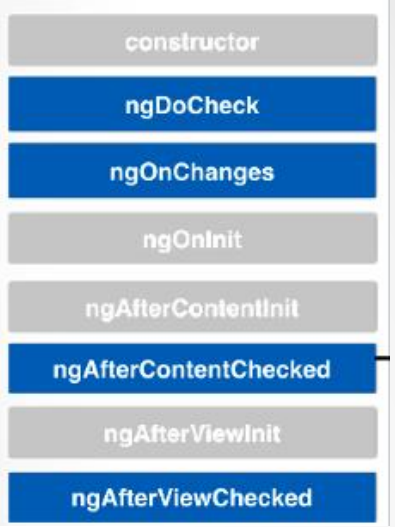
Various **events** happen during the **lifecycle** of an Angular component:

it gets **created**, **reacts to different events**, and gets **destroyed**

COMPONENT LIFECYCLE



COMPONENT LIFECYCLE

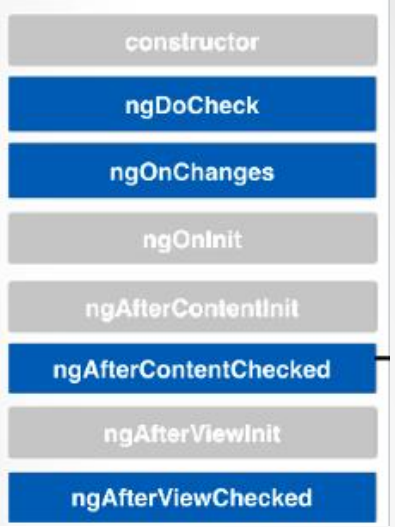


ngOnChanges() :

Called when a **parent component modifies** (or initializes) the values bound to the **input properties** of a child.

If the component has **no input properties**, ngOnChanges() isn't invoked.

COMPONENT LIFECYCLE

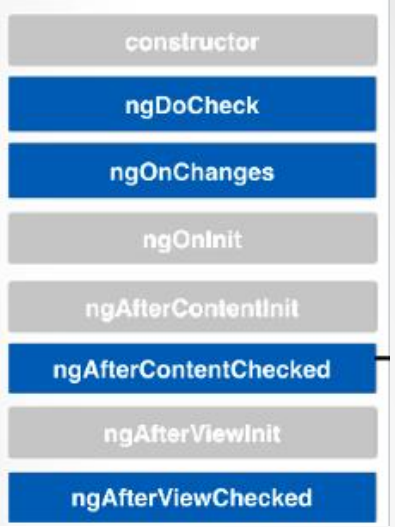


ngOnInit() :

Invoked **after the first invocation of ngOnChanges**.

By the time ngOnInit() is invoked, the component **properties will have been initialized**

COMPONENT LIFECYCLE

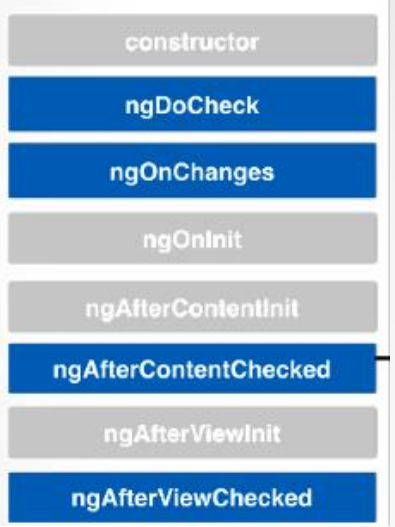


ngDoCheck():

Called on **each pass of the change detector**.

If you wish to implement a **custom change-detection algorithm** or add some debug code, write it in ngDoCheck()

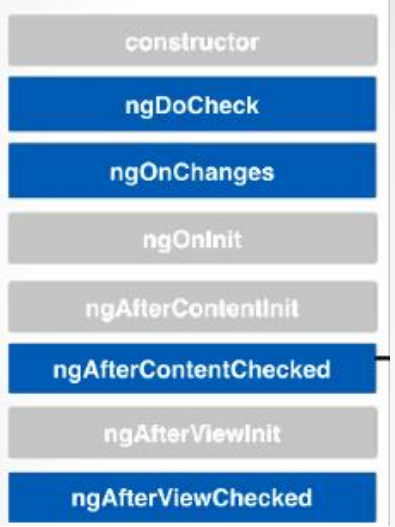
COMPONENT LIFECYCLE



ngAfterContentInit():

Invoked when the child component's **state is initialized** and **the projection completes**.

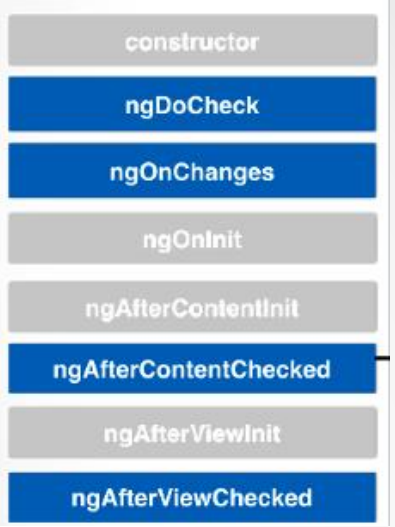
COMPONENT LIFECYCLE



ngAfterContentChecked():

During the **change-detection cycle**, this method is invoked on the component that has **<ng-content>**

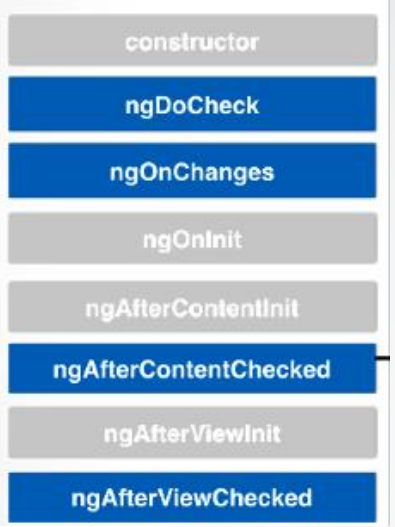
COMPONENT LIFECYCLE



ngAfterViewInit():

Invoked after a component's **view has been fully initialized**

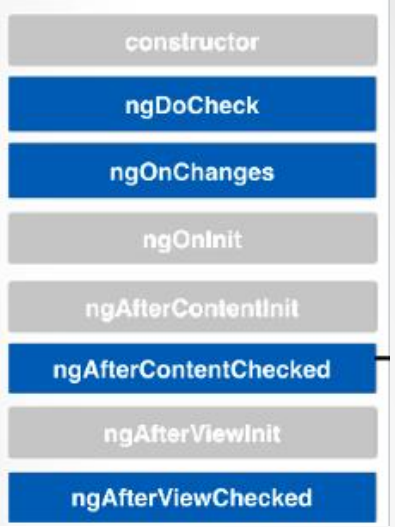
COMPONENT LIFECYCLE



ngAfterViewChecked():

Invoked when the **change-detection** mechanism checks whether there are **any changes in the component template's bindings**

COMPONENT LIFECYCLE



ngOnDestroy():

Invoked when the component **is being destroyed**

Code ...

Pipes & Directives

Pipes and Directives

We will take our components to the next level using Angular **pipes** and **directives**

Pipes allow us to digest and transform the information we bind in our templates

Directives enable more ambitious functionalities, such as manipulating the DOM or altering the appearance and behavior of HTML elements

Pipes and Directives

Manipulating data with pipes

Building pipes

Building directives

Pipes and Directives - Manipulating data with pipes

Pipes allow us to transform the outcome of our expressions at the view level

take data as input, transform it into the desired format, and display the output in the template

EX: expression | pipe

EX (params): expression | pipe:param

Pipes and Directives - Manipulating data with pipes

Pipes can be used with interpolation and property binding:

uppercase/lowercase: This transforms a string into uppercase or lowercase letters.

percent: This formats a number as a percentage.

date: This formats a date or a string in a particular date format.

currency: This formats a number as a local currency.

Pipes and Directives - Manipulating data with pipes

Pipes can be used with interpolation and property binding:

json: This takes an object as an input and outputs it in JSON format, replacing single quotes with double quotes.

keyvalue: This converts an object into a collection of key-value pairs

slice: This subtracts a subset (slice) of a collection or string

async: This is used when we manage data handled asynchronously by our component class

Code …

Pipes and Directives - Building directives

Angular directives are HTML attributes that extend the behavior or the appearance of a standard HTML element

When we apply a directive to an HTML element or even an Angular component, we can add custom behavior or alter its appearance.

Pipes and Directives - Building directives

There are **three types of directives**:

Components: Components are directives that contain an associated HTML template.

Structural directives: These add or remove elements from the DOM.

Attribute directives: These modify the appearance of a DOM element or define a custom behavior.

Code …