FLAG

Moder Angular

João Gonçalves

Interactive Developer / Trainer / Consultant

Applied Mathematics Degree

Master Multimedia Communication

http://joaogoncalves.net

edu@joaogoncalves.net

@joaopapin

FLAG

You ?

Partilha

https://github.com/JoaoGoncalves/flag-atangola

https://dontpad.com/atangola

FLAG

Intro

- Getting a **general overview** of common issues present in Angular apps from previous versions
- Learning about **new solutions for those problems provided in recent versions** of Angular

FLAG

ES6

ECMAScript

ECMAScript is a standard for client-side scripting languages

FLAG

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)

FLAG

ES6

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

FLAG

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
```

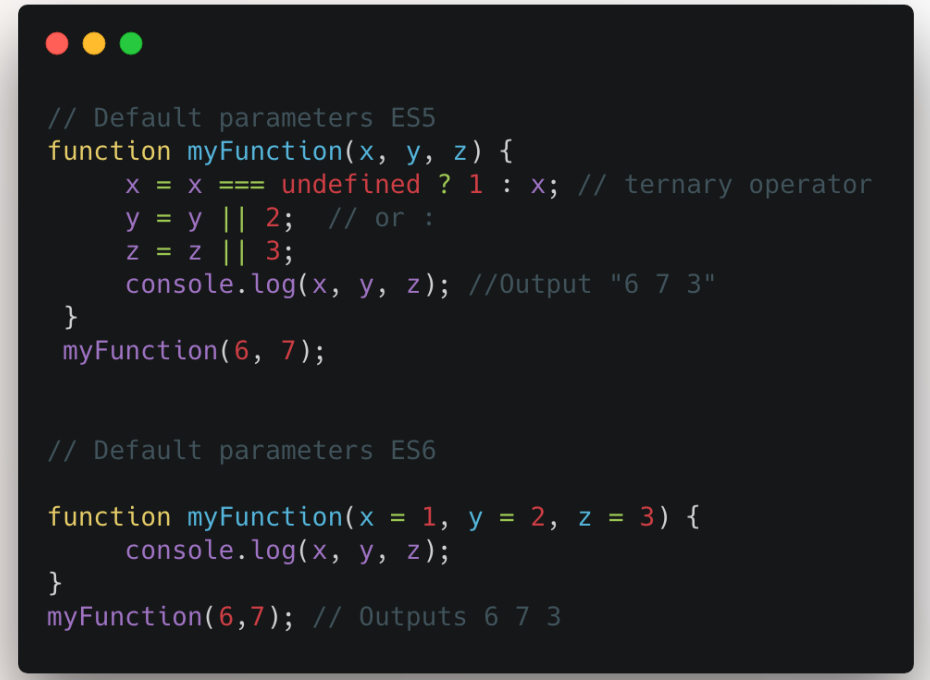
Redeclaring variables

```
// 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)

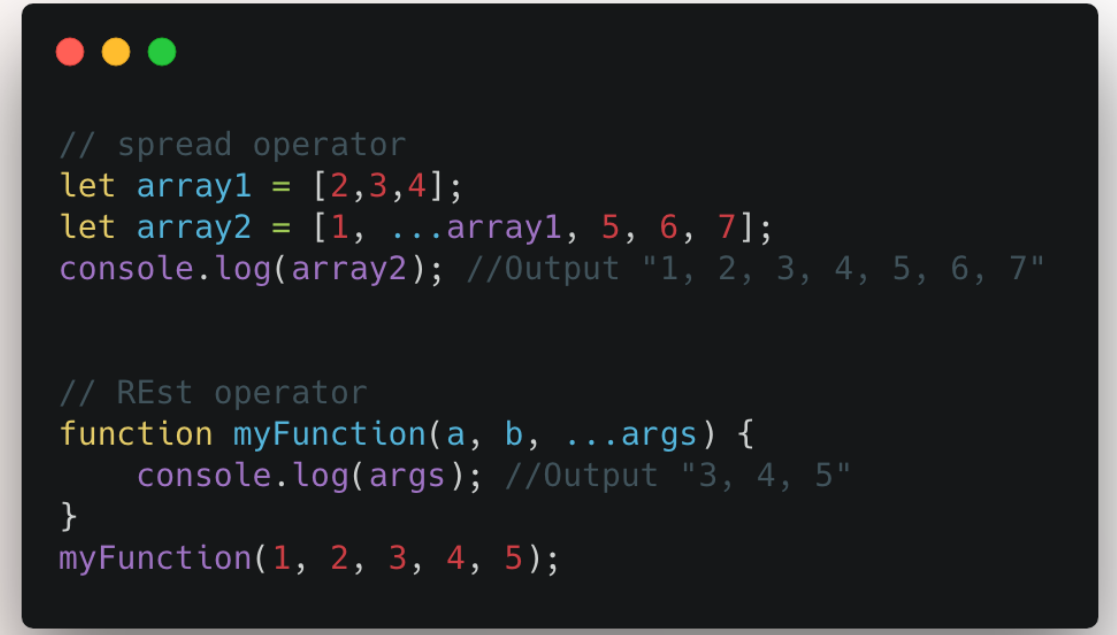
```
 // const

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



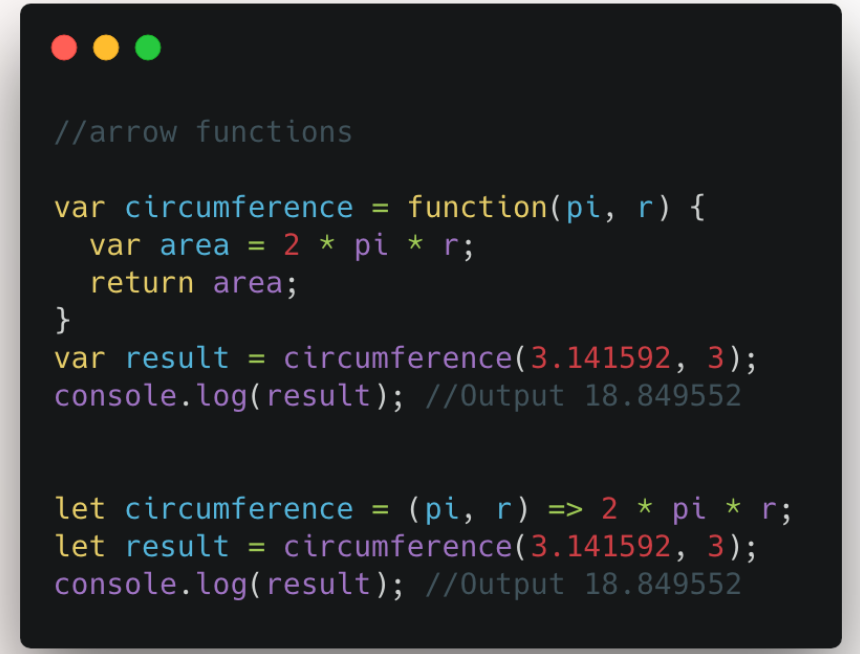
```
// spread operator
let array1 = [2,3,4];
let array2 = [1, ...array1, 5, 6, 7];
console.log(array2); //Output "1, 2, 3, 4, 5, 6, 7"


// 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
```

FLAG

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/

FLAG

Preparação Ambiente de trabalho

Instalar **Node.js e GIT**

Instalar **Typescript**:
(npm install -g typescript)

Instalar **Angular CLI**:
(npm install –g @angular/cli)

VSCode com extensões essential or extension pack

Debugger para o Chrome

FLAG

History…

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

Developed and maintained by Google

“Superheroic JavaScript **MVW** Framework”

FLAG

History…

Angular (2016)

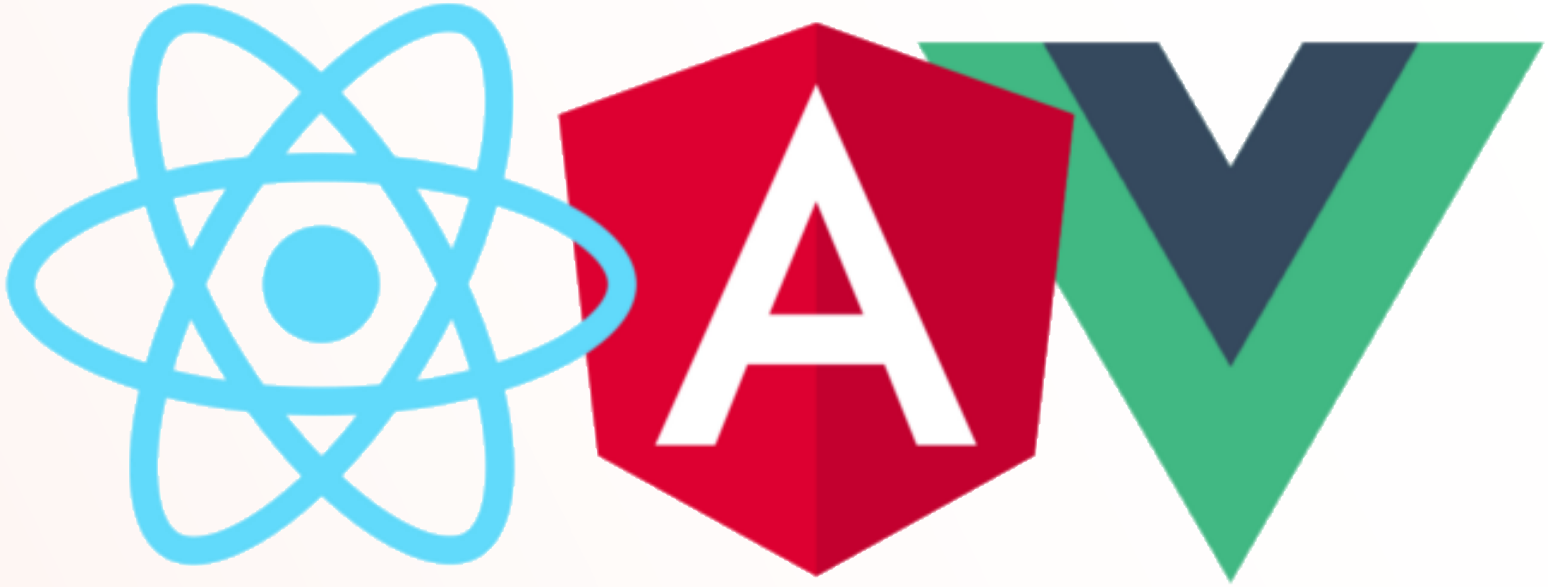
(also “Angular 2+”, “Angular 2” or “ng2”)

is the rewritten,

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

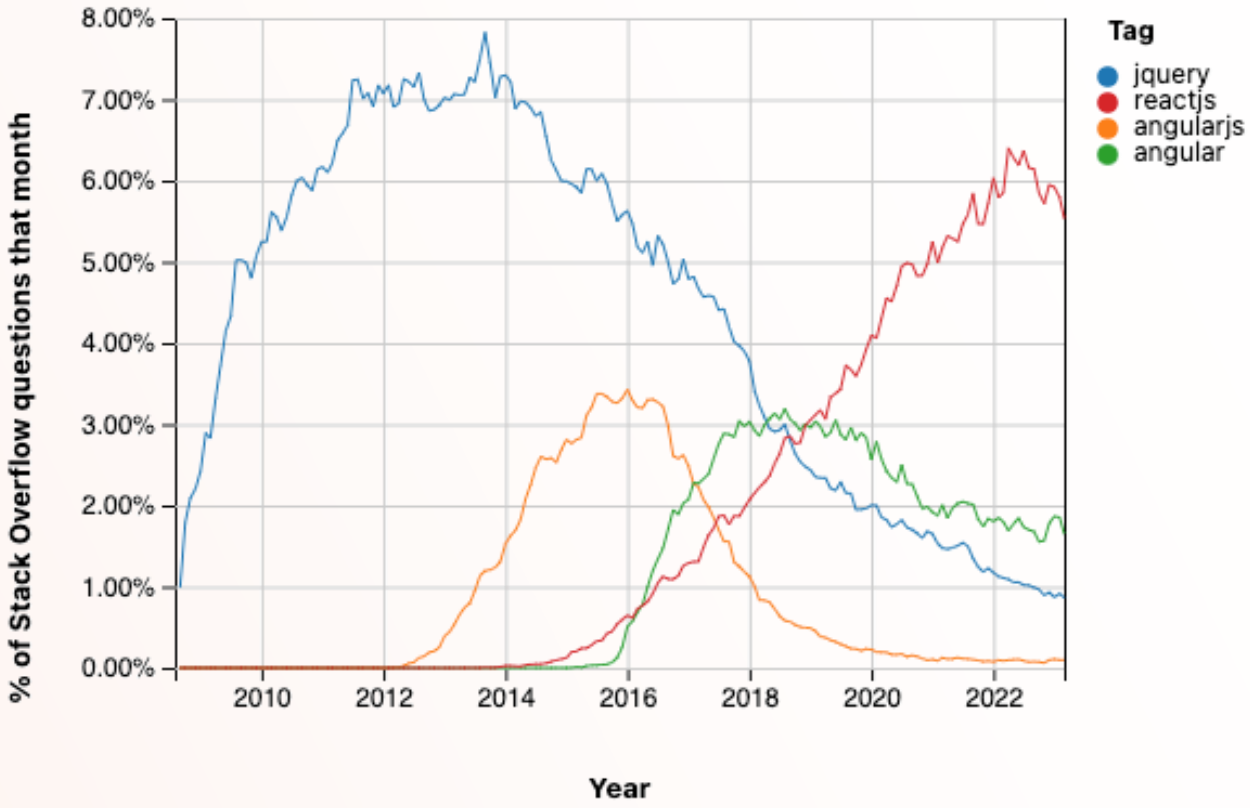
FLAG

History…

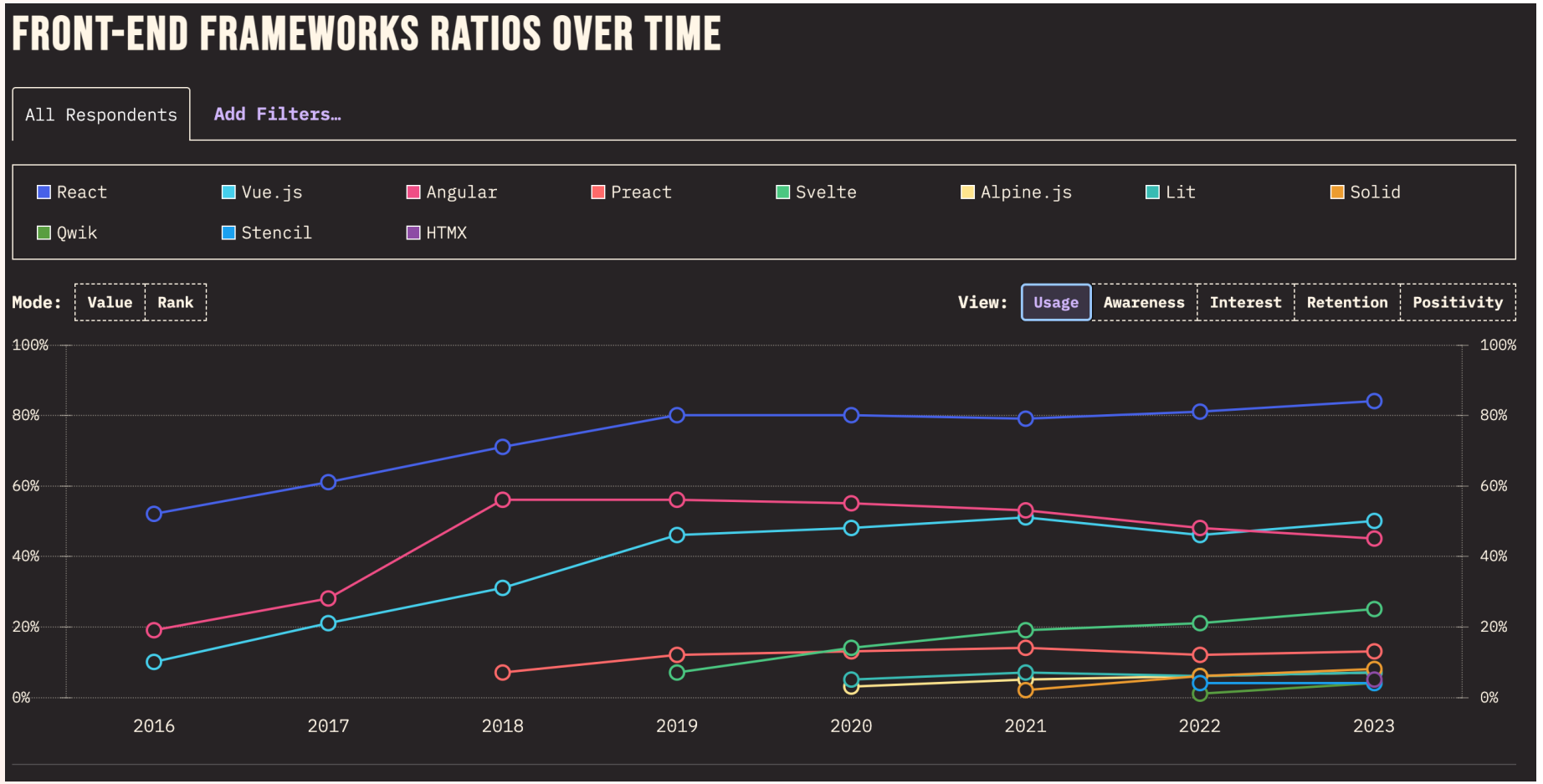


FLAG

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/

FLAG

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

FLAG

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)**

FLAG

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**

FLAG

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

FLAG

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

FLAG

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**

FLAG

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

FLAG

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

FLAG

Modern Angular

How the training will be Stuctured

follows a certain pattern of explaining these new features:

- **first**, we establish a problem that Angular developers experienced in previous versions, and cover solutions that the framework offered back in earlier versions
- **Second**, explain the new tools meant to solve the particular problem on a feature in a new, completely modern Angular application
- **Finally**, explain the ways in which developers can smoothly migrate their existing Angular applications to use the new feature

FLAG

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

FLAG

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**

FLAG

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”**

FLAG

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

FLAG

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**

FLAG

Modern Angular – Our Example

We Will create a application from scratch to discuss, all the solutions and response to questions like:

- What are the main parts of the app?
- How do those parts interact together and what serves as the glue between them?
- On an architectural level, what are the most important and frequent challenges developers face?
- How easy is it for someone to be onboarded into an existing project?

FLAG

Modern Angular – Our Example

HMRS (Human Ressources Management System):

Employees: all the data about a company's employees, their profiles, accessible both for the HR personnel and employees themselves

Recruitment: data about the recruitment process, interviews, and admissions, accessible to the HR personnel and certain employees

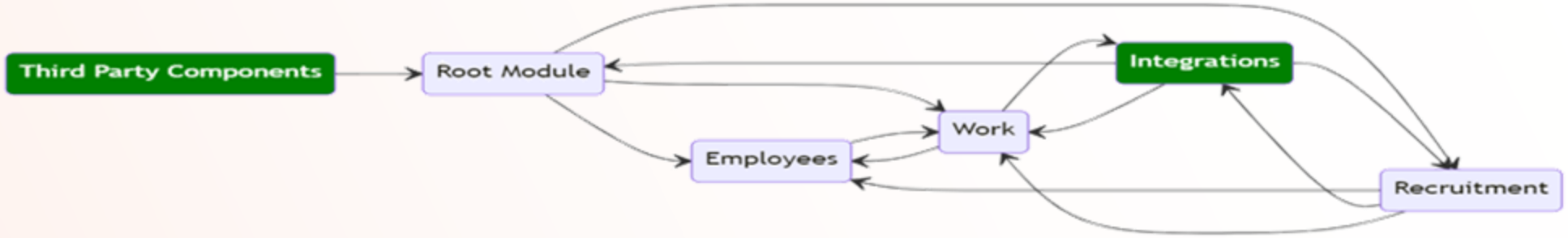
Time Off: a feature where employees can request time-offs and managers can approve them. Accessible to everyone

Work: data about the projects the company is working on, who reports to whom, submitting feedback, and so on.

Integrations: communications with third-party apps, like email or calendar.

FLAG

Modern Angular – HMRS



Modern Angular - HMRS

First, we will begin building this very application from scratch.

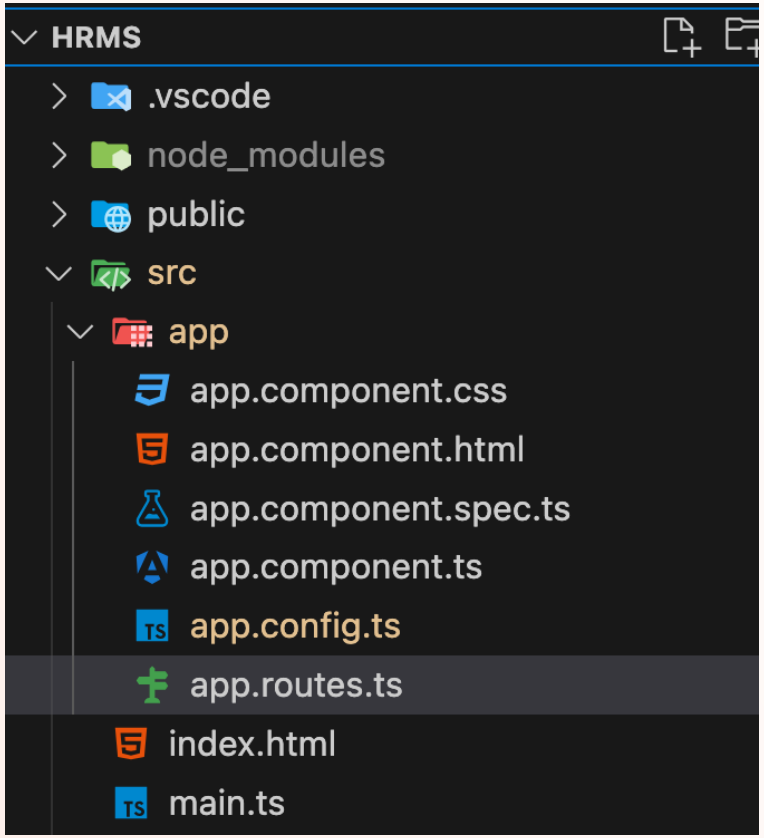
Next, in every session, we will examine scenarios in which this app already exists on an older version of Angular, so we can understand how to migrate it to use the latest features.

let us begin building this application by doing a basic setup

ng new hrms --defaults

FLAG

Modern Angular – Structure Changes

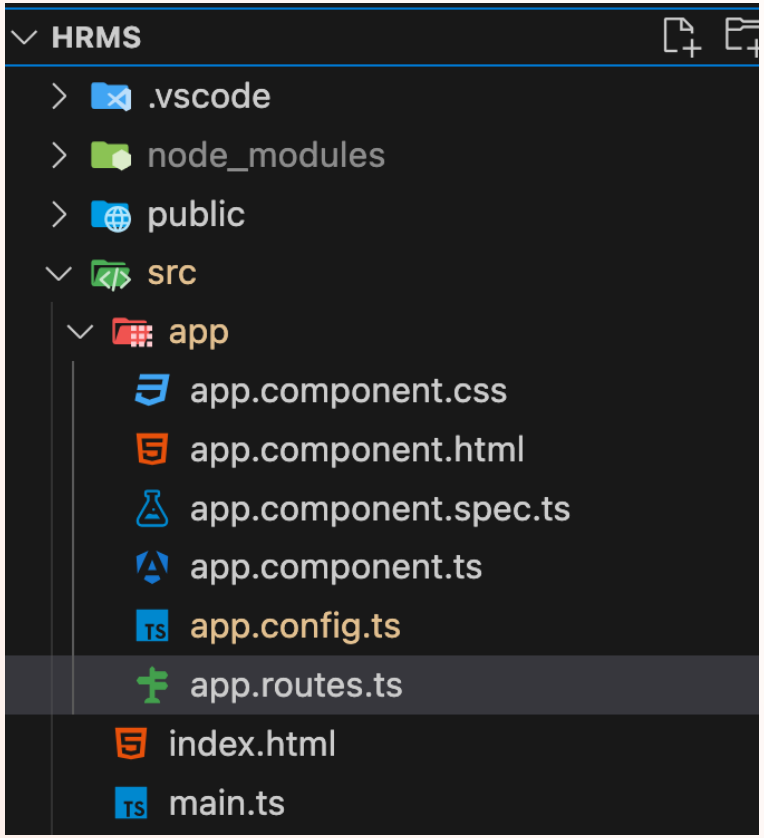


No "environments" folder: from Angular v15, environment files are not generated by default and can be added via a separate command

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



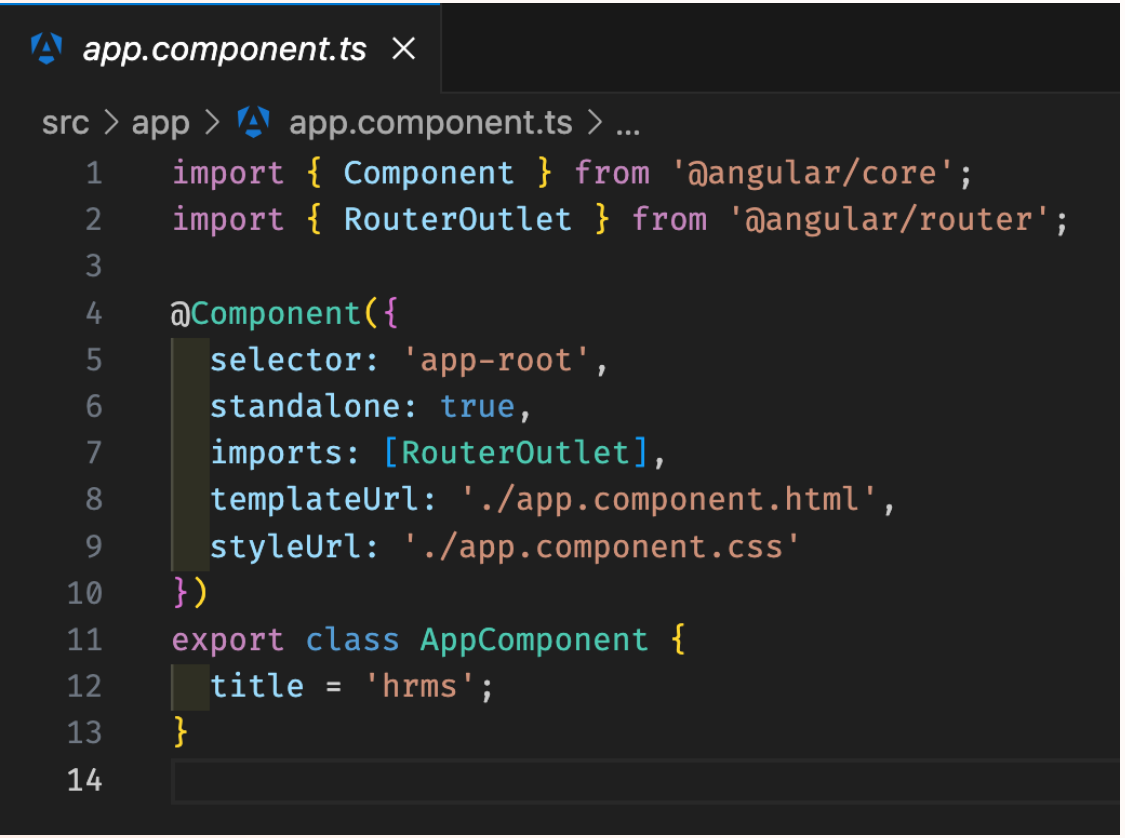
No app.module.ts file: The application is fully standalone and does not utilize modules for its architecture

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

FLAG

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

FLAG

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 > ...

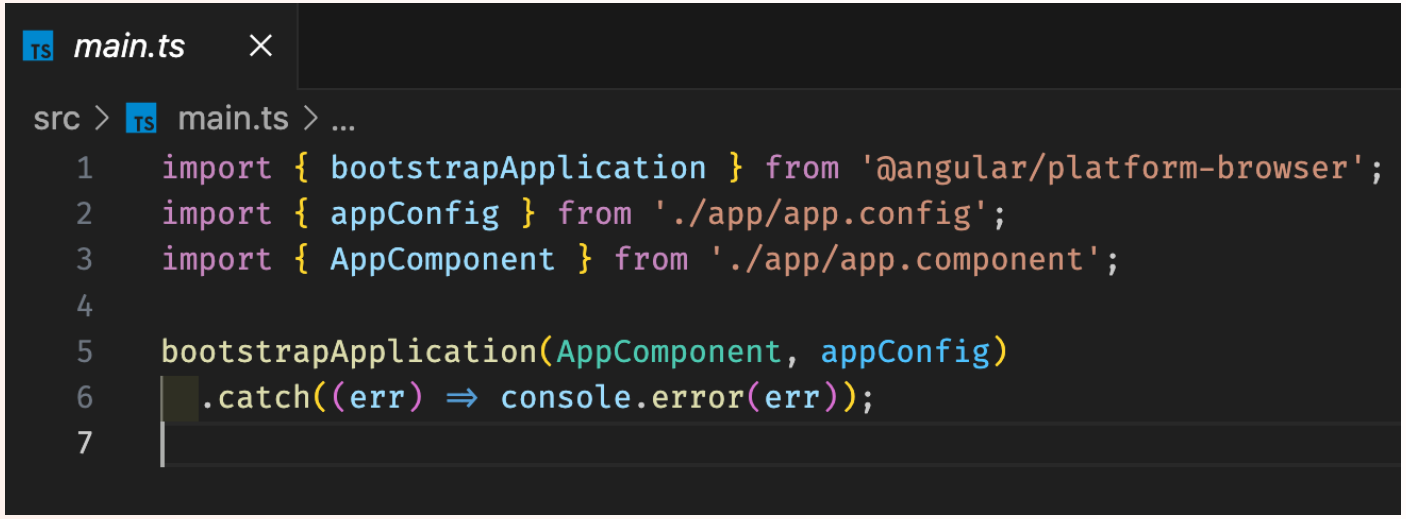
```ts
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

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

FLAG

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.

FLAG

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

FLAG

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**

FLAG

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**

FLAG

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

FLAG

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

FLAG

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

FLAG

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

FLAG

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)

FLAG

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

FLAG

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

FLAG

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

Migrating existing applications to standalone

FLAG

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

FLAG

Modern Angular – standalone future

Why abandon NgModules?

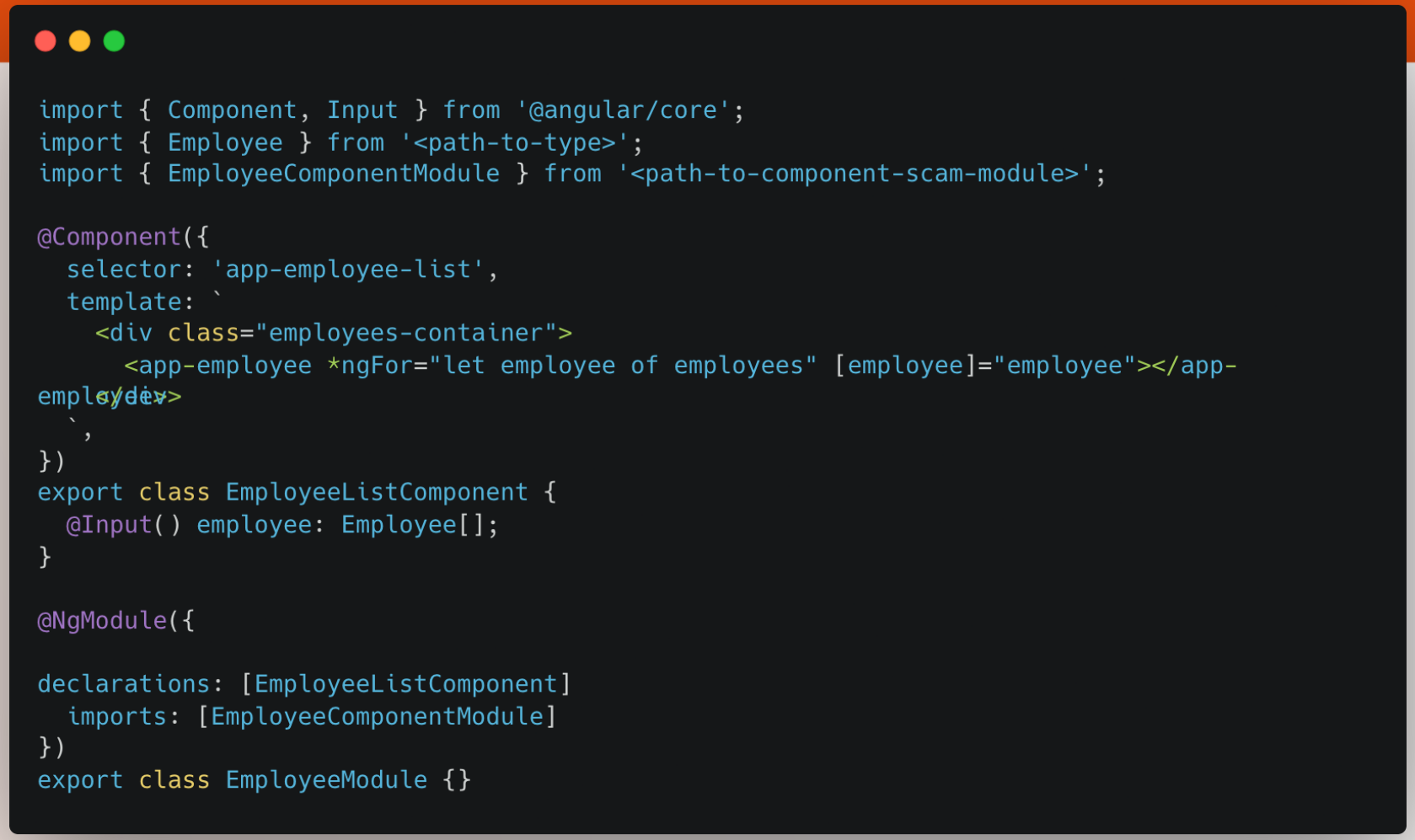
Hard to learn, hard to explain

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

Indirectness and boilerplate

FLAG

Modern Angular – standalone future



```
import { Component, Input } from '@angular/core';
import { Employee } from '<path-to-type>';
import { EmployeeComponentModule } from '<path-to-component-scam-module>';

@Component({
  selector: 'app-employee-list',
  template: `
    <div class="employees-container">
      <app-employee *ngFor="let employee of employees" [employee]="employee"></app-
employee>
    </div>
  `,
})
export class EmployeeListComponent {
  @Input() employee: Employee[];
}

@NgModule({

declarations: [EmployeeListComponent]
  imports: [EmployeeComponentModule]
})
export class EmployeeModule {}
```

SCAMs, (Single Component Angular Modules)

is an approach to building application structures based on the principle of having a single module for every single declarable

FLAG

Modern Angular – standalone future

SCAMs Benefit's:

Easier to track dependencies: they are listed in the same file

Easier to refactor and move around an application: we can just grab this file and move anywhere else

Easier to unit-test: we only have to mock the direct dependencies of this component

Better code splitting: we can lazy load components themselves directly

Easier to migrate to standalone

FLAG

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

FLAG

Modern Angular – standalone future

Creating our first standalone component

Creating our Login Component…

Code (At least…) ☺

FLAG

Modern Angular – standalone future

Routing standalone components and providing dependencies

Injecting a dependency in Angular comes down to defining a “token” of something, and telling the framework we want it somewhere

A provider is a place where we say “Dear Angular, if you see this token (for instance, AuthService as in our example) please inject this value (that particular instance)”

FLAG

Modern Angular – standalone future

QUESTION	With NgModules	Stand alone API's
Do I need to provide every dependency in every component?	Either this or mark the service as providedIn: 'root'	use a standalone API if one is provided
How do I import services that are provided in other modules	Import the module directly in the component	importProvidersFrom function in the application "main.ts"
How do I import built-in Angular dependencies?	Import the whole module	Use existing standalone APIs

FLAG

Modern Angular – standalone future

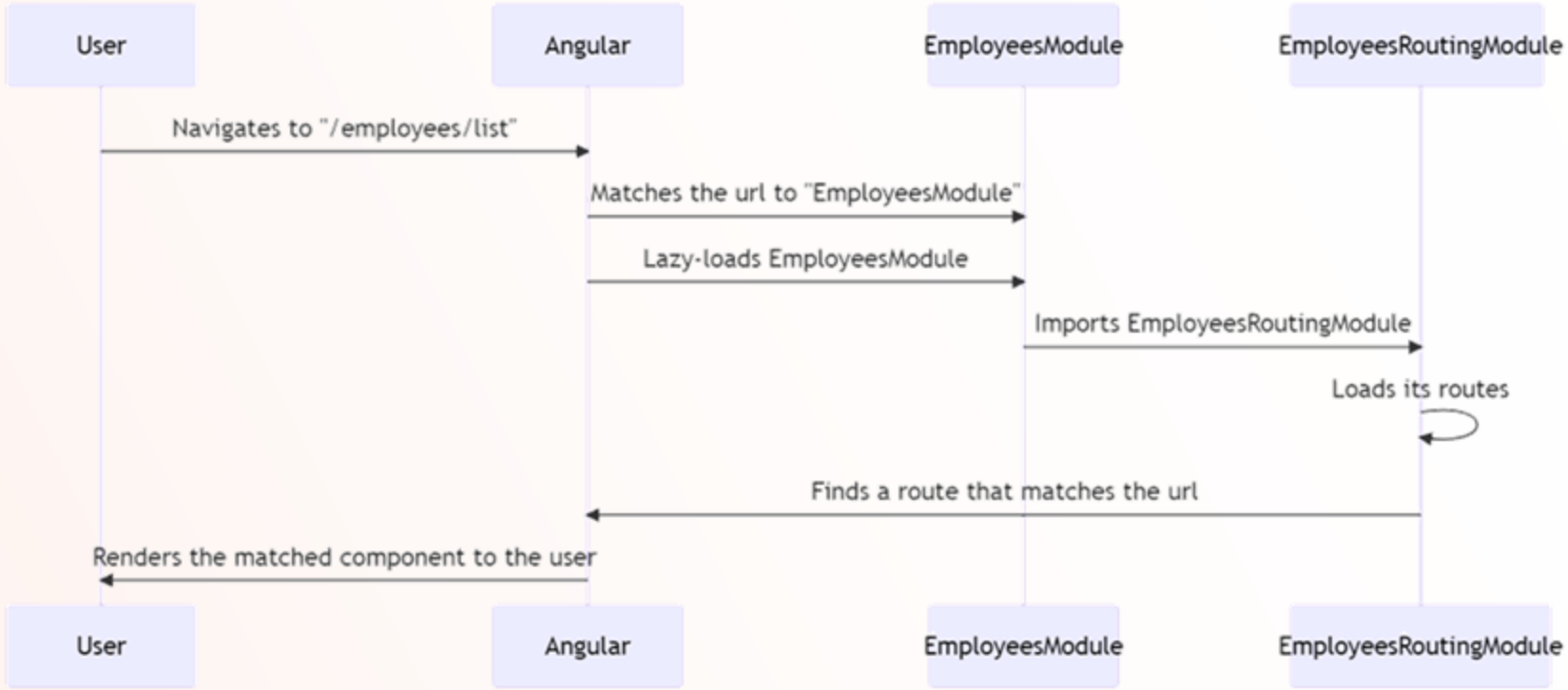
Back to Code …

FLAG

Modern Angular – standalone future

Lazy Loading Components

(with Modules)

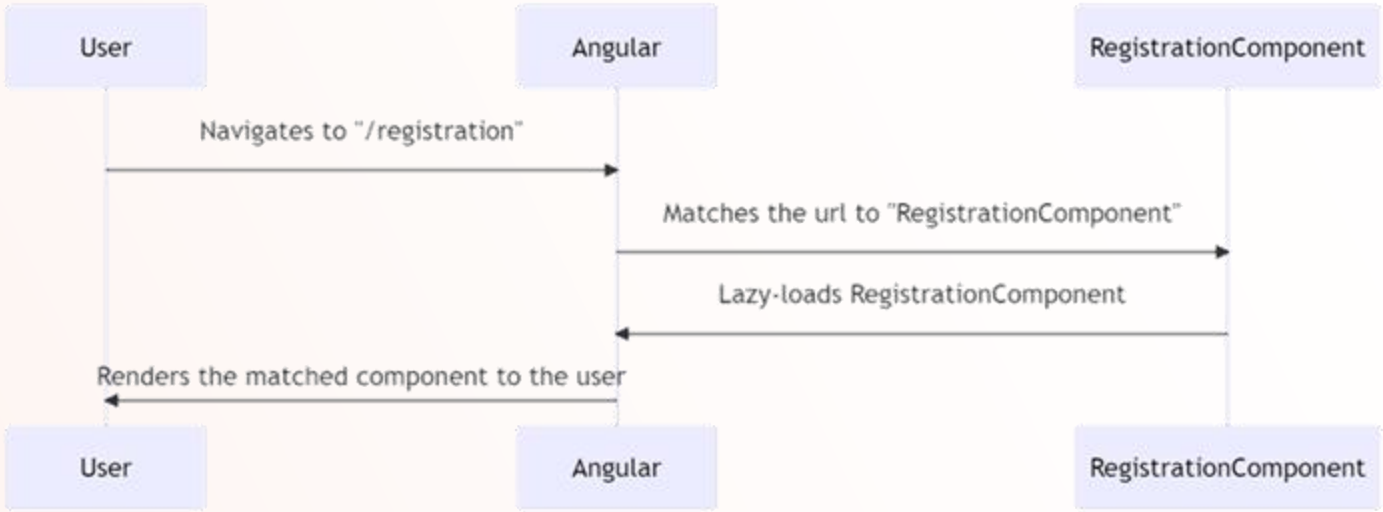


FLAG

Modern Angular – standalone future

Lazy Loading Components

(with stand alone component)



Modern Angular – standalone future

Back to Code …

FLAG

Modern Angular – standalone future

Lazy Loading Several Stand Alone Components

Can had several components that logically belong together
(e.g. EmployeeList, EmployeeDetails, CreateEmployee, and EditEmployee)
can be easily lazy-loaded together

we also might want our folder structure to better reflect the hierarchies that exist within both our routing and the application in general

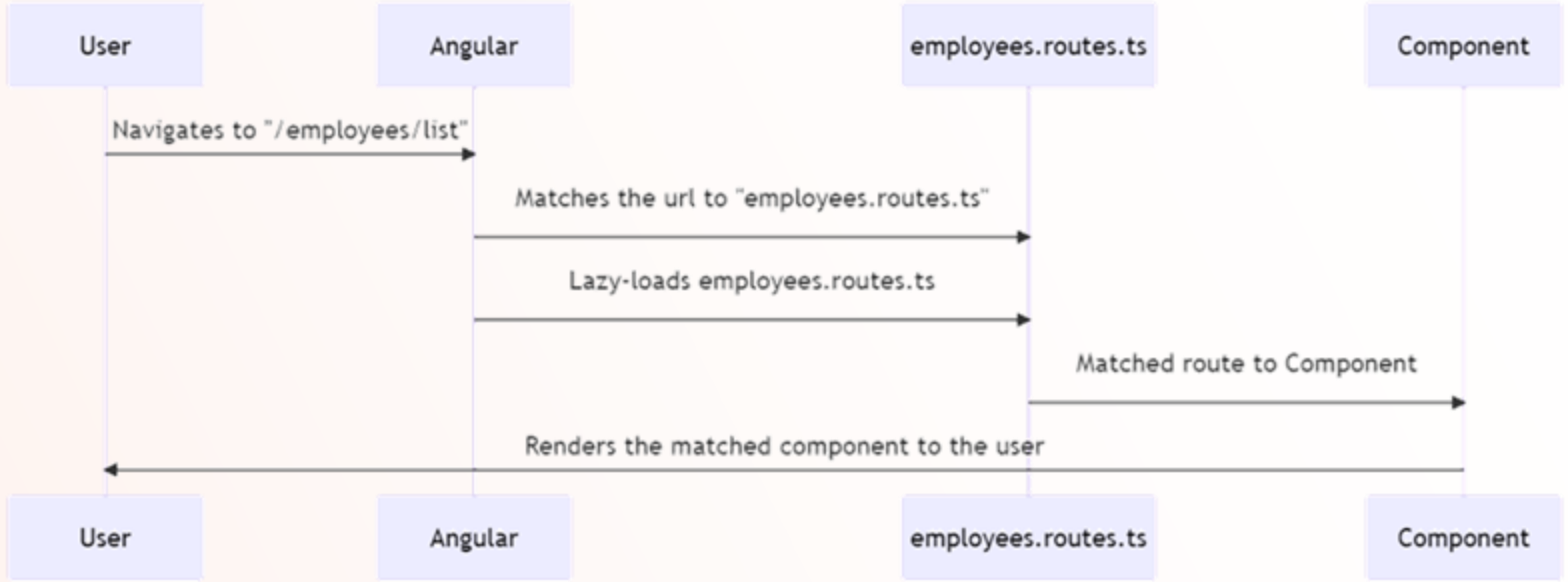
Code …. Create the components first…

FLAG

Modern Angular – standalone future

Lazy Loading Several Stand Alone Components

the only key difference as compared to NgModule setup here is that the loadChildren function just imports the route



FLAG

Modern Angular – standalone future

Providing dependencies only to certain routes

(it is now possible to add a providers array to a route definition)

let's create an EmployeeService in the "src/app/services" directory, and put some code in it responsible for HTTP calls

Inject this service only to "Employees" Route

Code …

FLAG

Modern Angular – standalone future

Lazy-loading a component into another component

Code …

FLAG

Modern Angular – standalone future

Migrations and common pitfalls

But what if we want to entirely migrate existing applications to fully use standalone?

Let’s examine some approaches we can take

FLAG

Modern Angular – standalone future

Migrations and common pitfalls

1
Migrating by hand

Adopt the rule of authoring all new components/directives/pipes as standalone

Choose components that we want to convert one by one,
and mark them as standalone: true

In the NgModule in which this component is declared,
move it from the declarations array to the imports array

FLAG

Migrations and common pitfalls

1
Migrating by hand

Remove NgModule-s one by one, until only AppModule remains

When only AppModule is left, remove it and move the providers to “main.ts” (appConfig)

Modern Angular – standalone future

Migrations and common pitfalls

2
Using SCAMs

adopt making all new building blocks standalone

Choose components that we like to transition first,
but instead of making them standalone, just create a SCAM module

start marking all components, directives,
and pipes as standalone and remove their SCAM module

FLAG

Modern Angular – standalone future

Migrations and common pitfalls

2
Using SCAMs

start removing feature modules and making their own declarable components/directives/pipes standalone

remove AppModule and use standalone provider API

FLAG

Modern Angular – standalone future

Migrations and common pitfalls

3
Migrating with a schematic command

Make sure the app we want to upgrade is on version 16

On terminal: *run ng g @angular/core:standalone*

Run, one by one: “Convert all components, directives, and pipes to standalone”, “Remove unnecessary NgModule classes”, and “Bootstrap the application using standalone APIs”

FLAG

Modern Angular – standalone future

Migrations and common pitfalls

Example…

FLAG

Modern Angular – standalone future

Summary

NgModule-s have outstanding issues like boilerplate, increased complexity, and a steep learning curve

Single Component Angular Modules (SCAMs) can mitigate those issues

From v14, standalone Angular components/directives/pipes are available, marked as standalone: true in their metadata

Applications can now be built completely without NgModule-s, using the special bootstrapApplication function in the “main.ts” file

FLAG

Modern Angular – standalone future

Summary

Standalone components/pipes/directives can import other standalone building blocks via an imports array in their metadata

Standalone building blocks can fully interop with NgModule-s by either importing the module directly into components or via the importProvidersFrom function

Lazy-loading routes is now possible for standalone components directly, instead of whole modules

Existing applications can migrate to standalone either by hand or using the schematic provided by the Angular team

FLAG

FLAG

www.flag.pt

LISBOA Edifício Mirage, R. Dr. Eduardo Neves, Nº3, 1050-077 Lisboa email: querosabermais@flag.pt Tel. 213 560 606

PORTO Rua Oliveira Monteiro, Nº 168, 4050–438 Porto email: querosabermais@porto.flag.pt Tel. 226 094 789