Modern angular

João Gonçalves

Interactive Developer / Trainer / Consultant

Applied Mathematics Degree

Master Multimedia Communication

http://joaogoncalves.net

edu@joaogoncalves.net

@joaopapin

You ?

Partilha

https://github.com/JoaoGoncalves/MoOngy-Angular

https://dontpad.com/moongy-angular

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

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

History…

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

Developed and maintained by Google

“Superheroic JavaScript **MVW** Framework”

History…

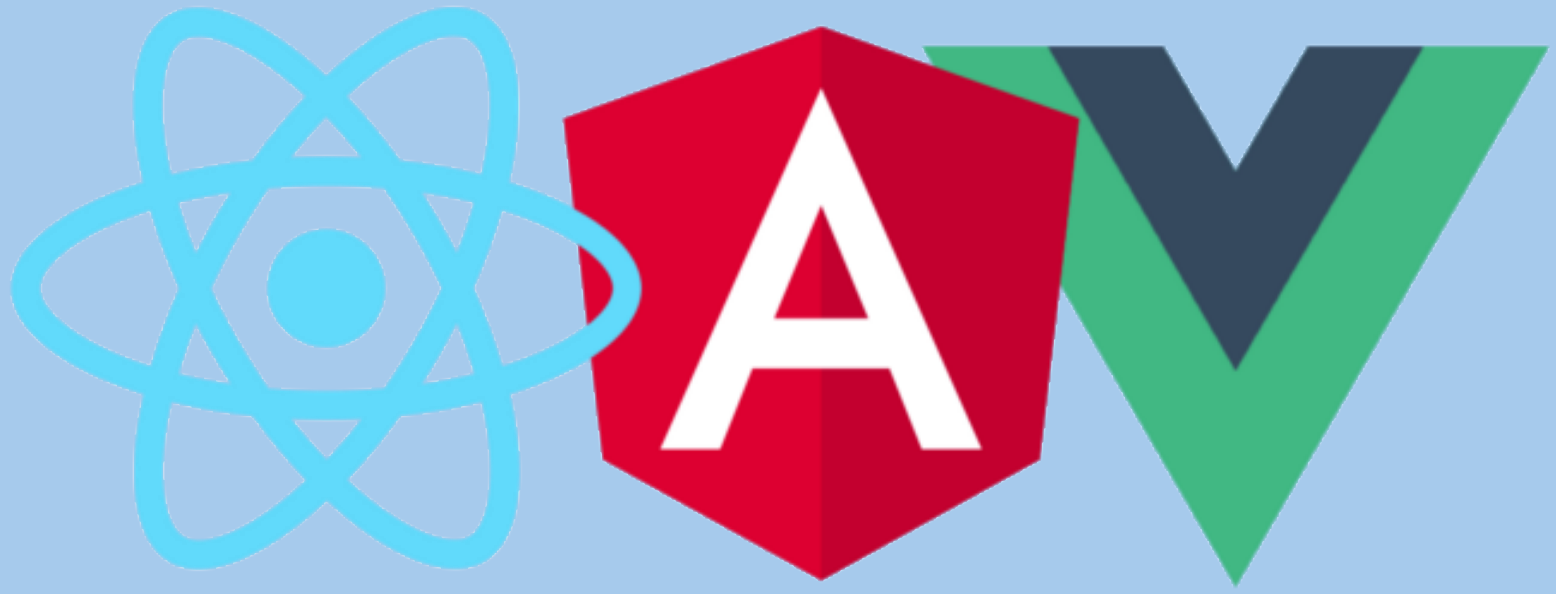
Angular (2016)

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

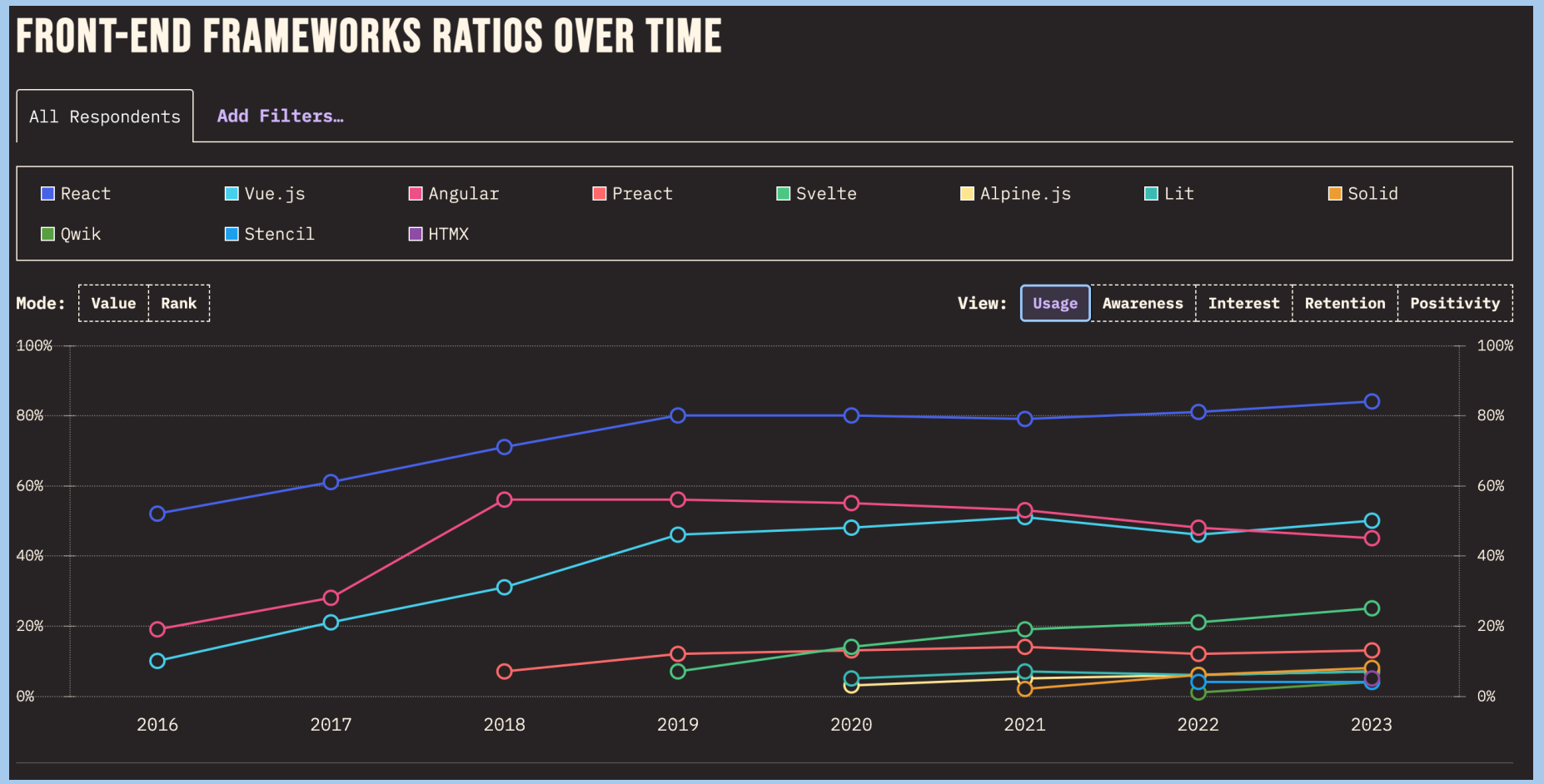
is the rewritten,

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

History…



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

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 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

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 – 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?

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

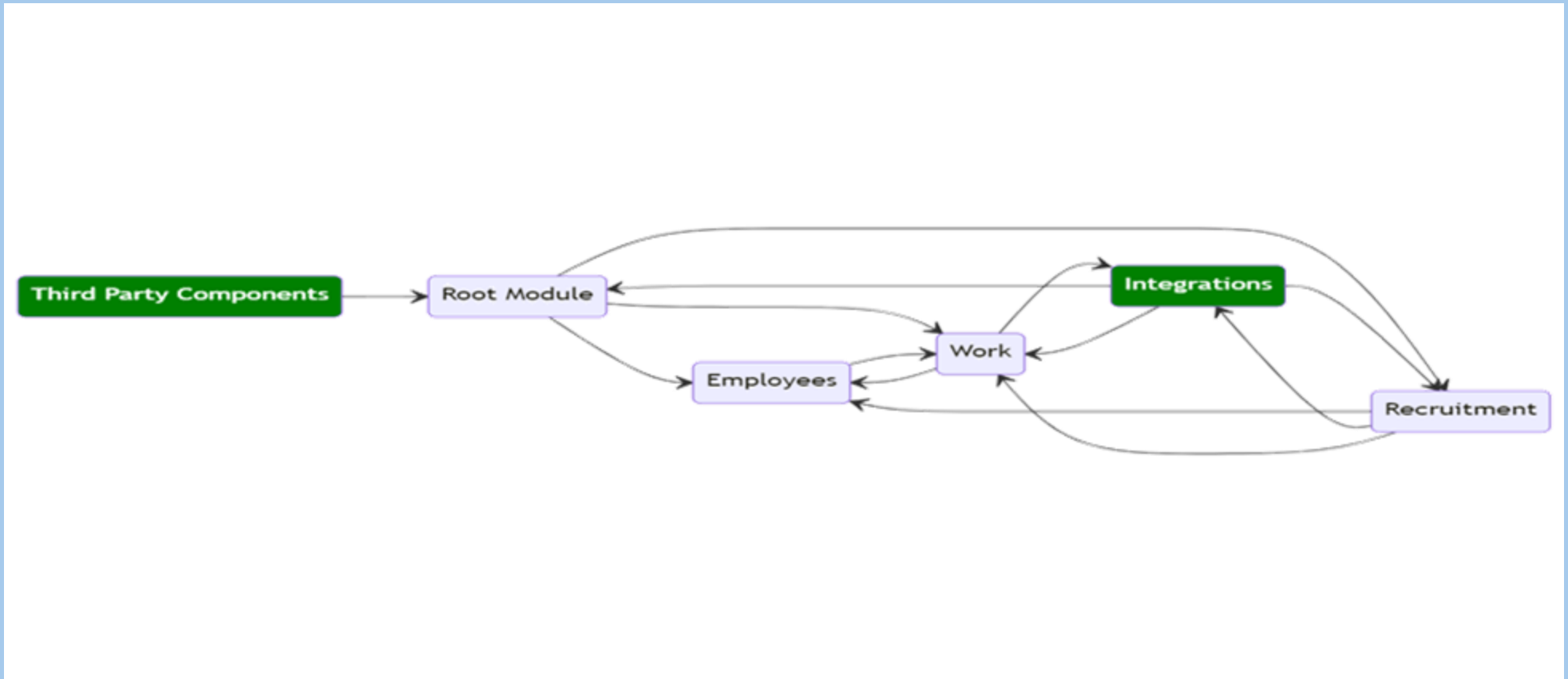
Recruitment: data about the recruitment process, interviews, and admissions, accessible to the HR personnel and certain employees

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.

Modern Angular – HMRS



Modern Angular - HMRS

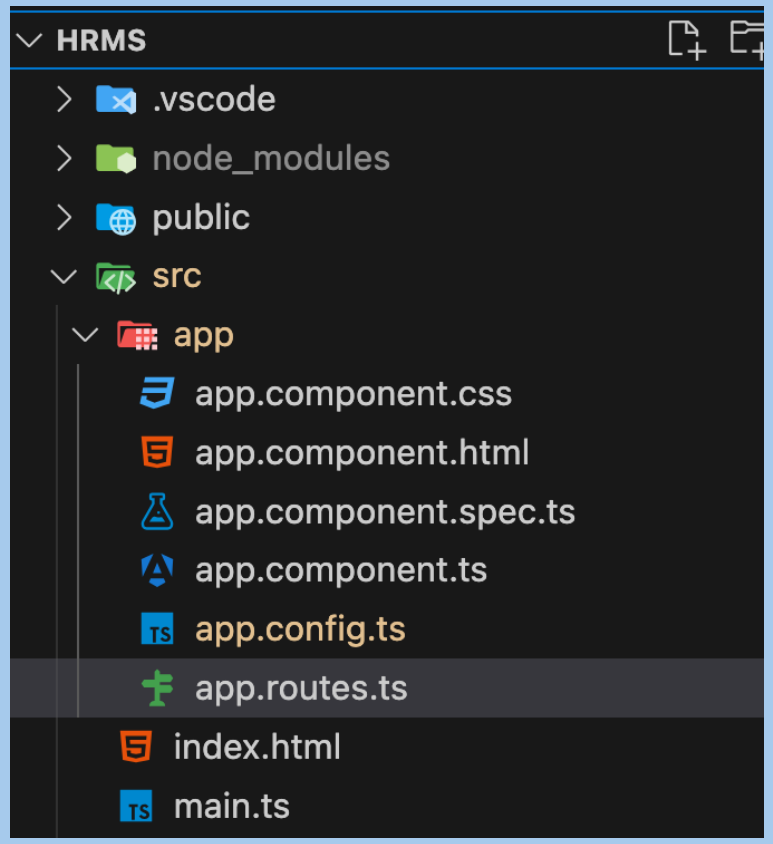
First, we will begin building this very application from scratch.

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

Modern Angular – Structure Changes

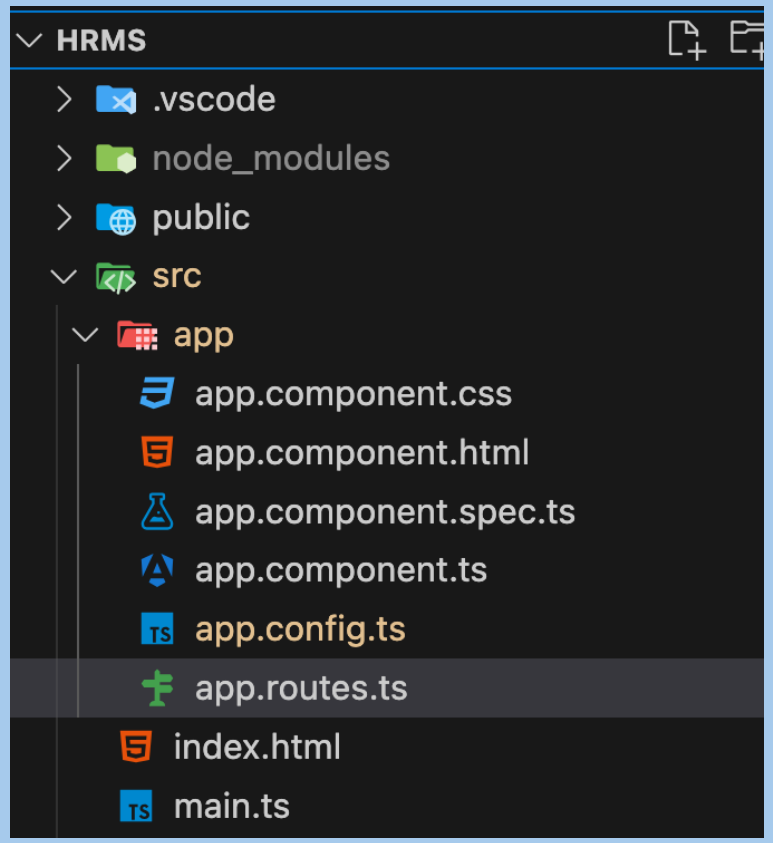


No "environments" folder: from Angular v15, environment files are not generated by default and can be added via a separate command

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

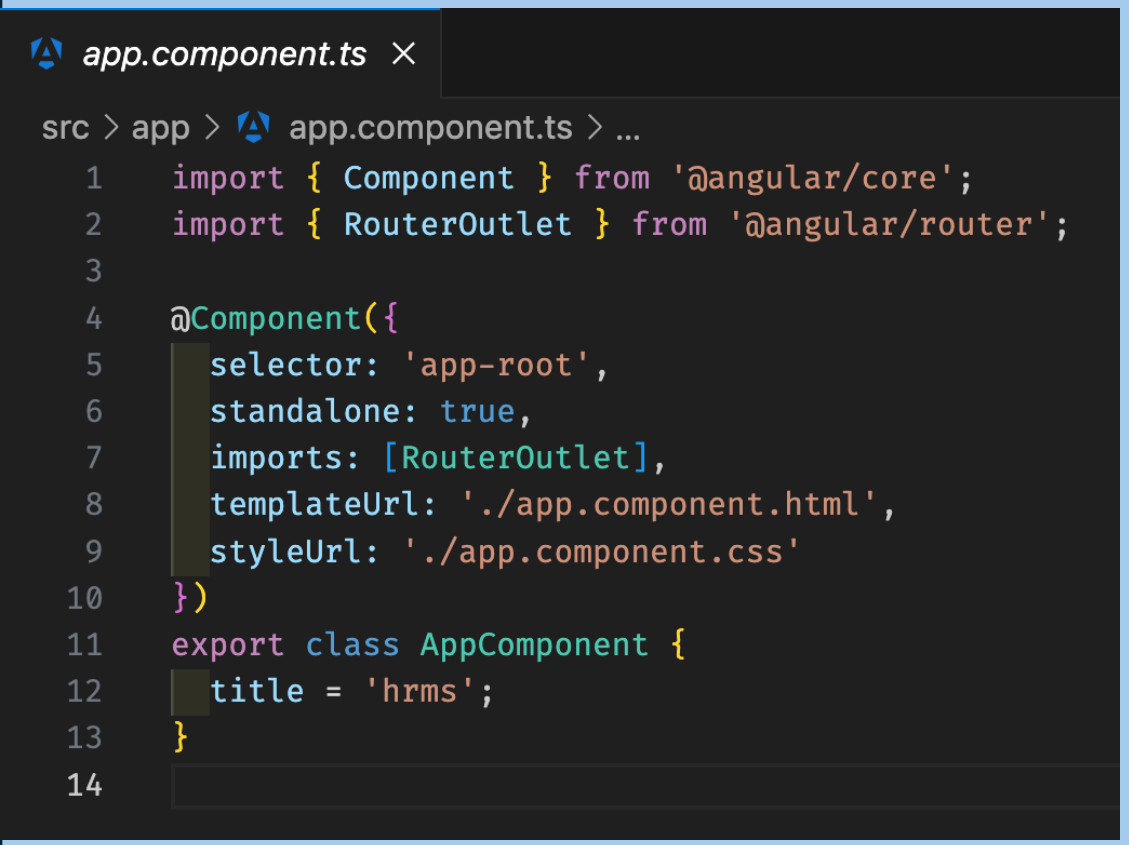


No app.module.ts file: The application is fully standalone and does not utilize modules for its architecture

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

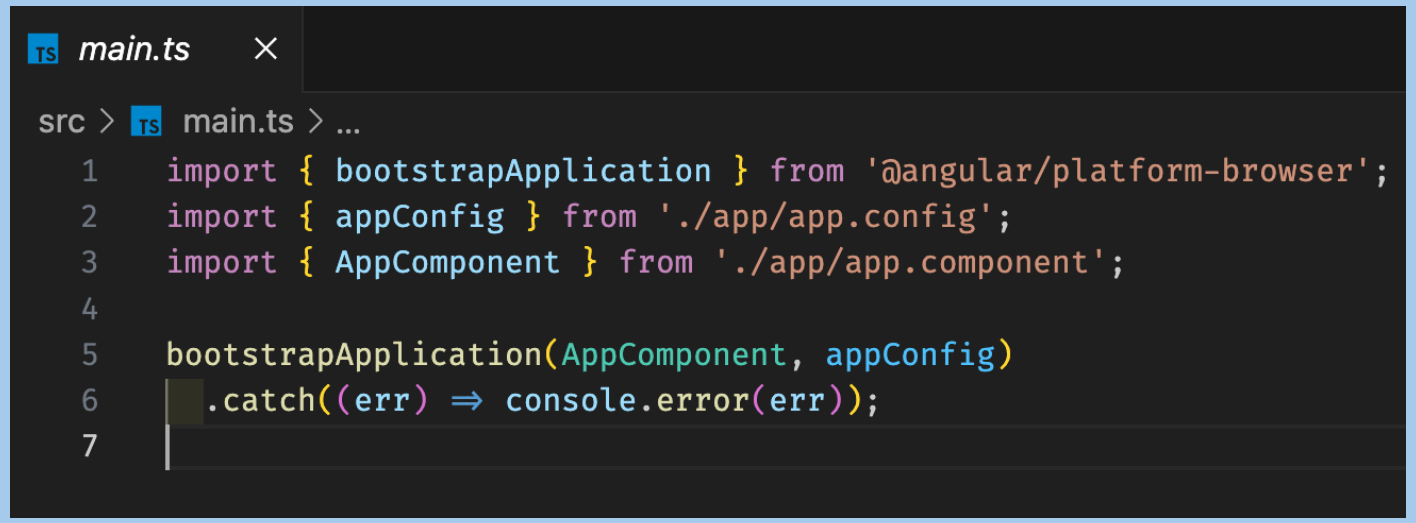
src > app > app.config.ts > ...

```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

Migrating existing applications to standalone

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

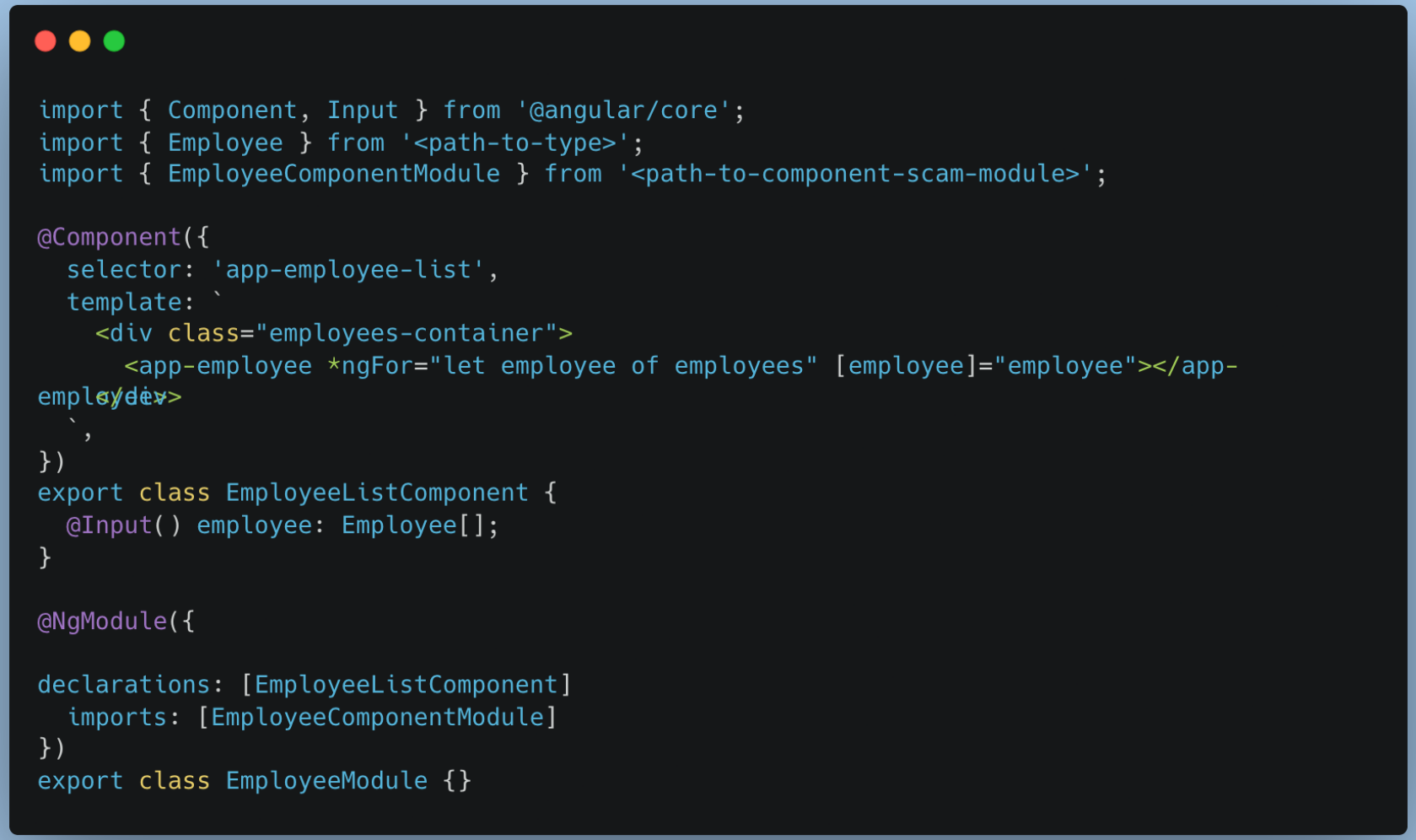
Why abandon NgModules?

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



```
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

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

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

Modern Angular – standalone future

Creating our first standalone component

Creating our Login Component…

Code (At least…) ☺

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)”

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

Modern Angular – standalone future

Back to Code …