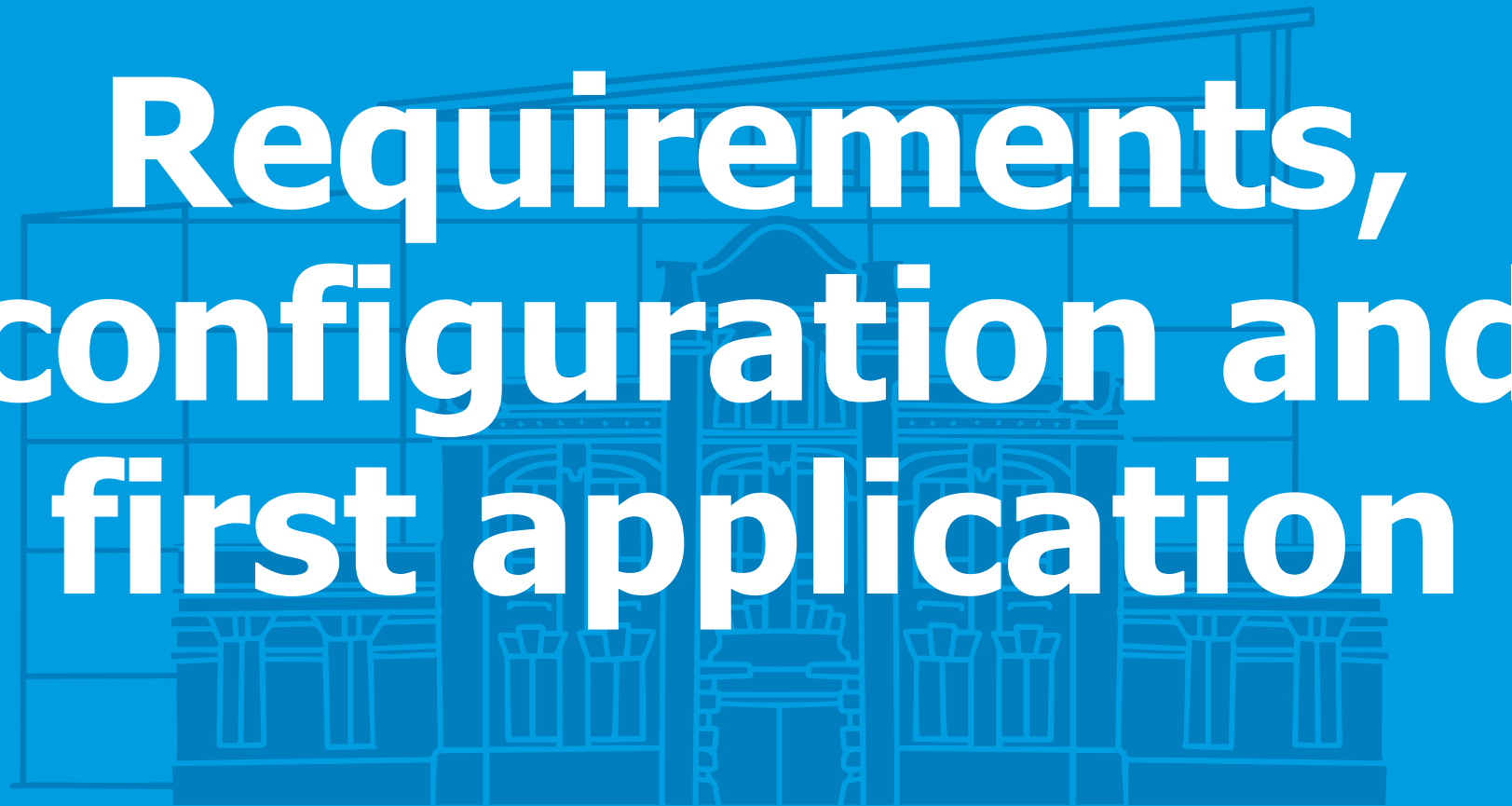
netmind
a bts company

Angular 14 - 04

Requirements, configuration and first application



Angular CLI

📎 It is a **command line interface** that allows us to generate projects and add components in an interactive and very simple way.
Some common commands:

```
ng new <project> : create a new project
ng serve: start the server
ng build [--prod/--dev --e=prod/dev]: build the package
ng g <component_type>
```

Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe/filtre	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-module

Angular CLI

🔗 Installing the CLI

```
npm install -g @angular/cli
```

🔗 Documentation

https://github.com/angular/angular-cli

🔗 Create a project

Move to your workspace folder:

```
ng new my-new-app
cd my-new-app
ng serve
```

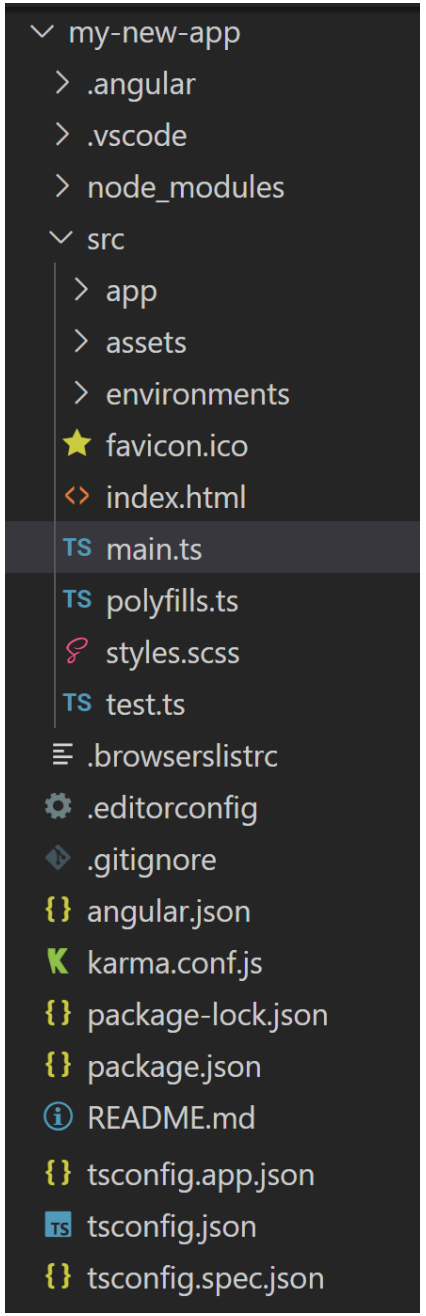
Open browser in http://localhost:4200/

Project structure

- **package.json**: project and dependency configuration
- **src**: directory with the modules and components of the application
  - main.ts: application entry point
  - index.html: SPA file
  - app: app code
  - assets: static files like imgs.
  - styles.scss: global styles
- **tsconfig.json**: ts configuration
- **angular.json**: primary configuration file for the project
- **karma.conf.js**: configuration for karma (test manager)
- **tslint.json**: ts code linter configuration (code quality)
- **protractor.config.js**: protractor configuration (tests end to end or how users see the app)

- **e2e**: directory with test specifications for protractor
- **src/environments**: directory with environment settings

https://angular.io/guide/file-structure



Entry point: main.ts

📎 **The entry point of an angular application is set in the main.ts file.**
The minimum parts required are these simple lines of code:

Note that the type of platform is indicated (there can be several), and, on it, the main module (AppModule) is configured.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Entry point: main.ts

🖇 **The last TypeScript block kicks off the Angular engine**

It must be configured for browsers, with **platformBrowserDynamic**, which returns an object prepared to do the "bootstrapping" or initial loading process of angular

This code is ready to run on any engine that supports it, on any platform.
In this case it sets a reference to a particular platform.

Entry module

🔗 **Every Angular application must have -at least- one module (the root module).**
A class becomes a module when it is **decorated** with the **NgModule** decorator.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Entry module

🔗 **Every Angular application must have -at least- one module (the root module).**
Here, an AppModule class is defined and exported for its use by other components.

In turn, that class imports @angular/core and @angular/platform-browser with the syntax that we have already seen in the TypeScript part.

The class contains a decorator (**@NgModule**) that declares the BrowserModule import for use by the class by referencing it in the imports attribute
This attribute defines an array of imported elements that will be used later.
In more realistic applications, other modules will be needed, such as FormsModule, RouterModule or HttpModule.

The **AppComponent** is imported and marked for bootstrapping.

Entry component

🔗 **Every Angular application has at least one component (the root component).**
By convention, we create that root component in the same folder, with the name **app.component.ts**.
Its content will be a component named **AppComponent**.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'my-new-app';
}
```

Entry component

🖇 The component imports the **@Component** decorator definition of "@angular/core".

Next define the component's UI within the decorator. This has two parts:

- A **directive** definition (what will be used in the HTML page)
- An **HTML template**, the templateUrl definition by pointing to a separate HTML snippet in a file (a view).
- A style source (SCSS in this case)

Lastly, the **class definition** itself, to which the decorator is associated (exported, to be used externally).

- Later it will carry its own business logic, but here its only function is to support the decorator.

Index.html (main page)

🔗 **The index.html page is the one that takes advantage of this functionality and displays the content.**

The necessary libraries will be injected and it will include a UI element that matches the one we have defined in our component: **<app-root>**Loading...**</app-root>**

In this version there are no references to libraries (which did appear in versions 2+), which are injected at "deployment" time by the development environment. That is part of the "bundling and minifying" that we have already mentioned

However, at runtime we will see those references

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyNewApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="icon" type="image/x-icon"
href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>MyNewApp</title>
        <base href="/">
        <meta name="viewport" content="width=device-width, initial-
scale=1">
        <link rel="icon" type="image/x-icon" href="favicon.ico">
        <link rel="stylesheet" href="styles.css">
    </head>
    <body>
        <app-root></app-root>
        <script src="runtime.js" type="module"></script>
        <script src="polyfills.js" type="module"></script>
        <script src="styles.js" defer></script>
        <script src="vendor.js" type="module"></script>
        <script src="main.js" type="module"></script>
    </body>
</html>
```

Chaining of loading processes

📎 All starts with the definition included in the **angular.json** file that sets the bootstrap module and which is the home page:

```
"projects": {
    "my-new-app": {
      "projectType": "application",
      "schematics": {
        "@schematics/angular:component": {
          "style": "scss"
        }
      },
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/my-new-app",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": "src/polyfills.ts",
            "tsConfig": "tsconfig.app.json",
            "inlineStyleLanguage": "scss",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/styles.scss"
            ],
            "scripts": []
          },
           ...
```

Chaining of loading processes

🔗 **Webpack** springs into action when calling commands
- ng build
- ng serve

There, a complex process of filtering and packing JavaScript, HTML and CSS fragments takes place.

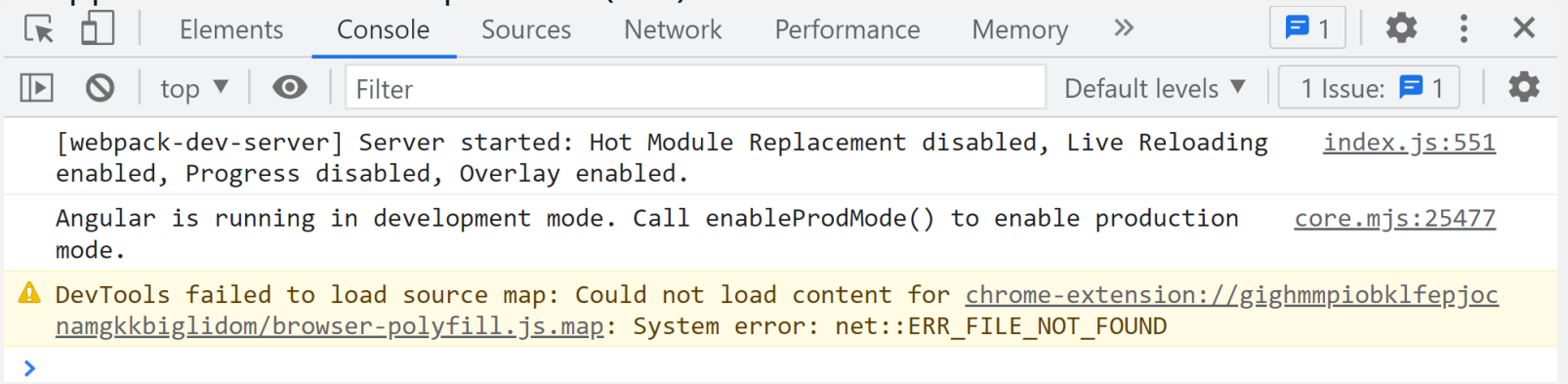
As **main.ts** in turn loads **AppModule** and registers it with the call to **platform.bootstrapModule (AppModule)**
...which launches every sequence.

When the entry file (index.html or other), has been "**injected**" with the necessary scripts, we can open the page in any browser on the indicated port (4200, by default).

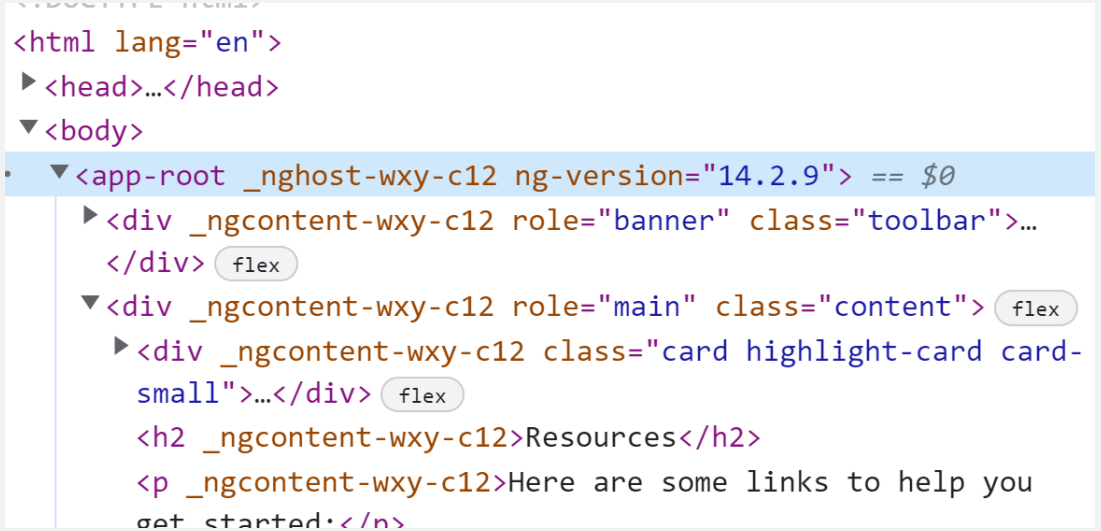
app.module, in turn, imports **app.component** (among other things), and it is **app.component** that defines the user interface.

Debugging

We can see what happens with the developer tools (**F12**):

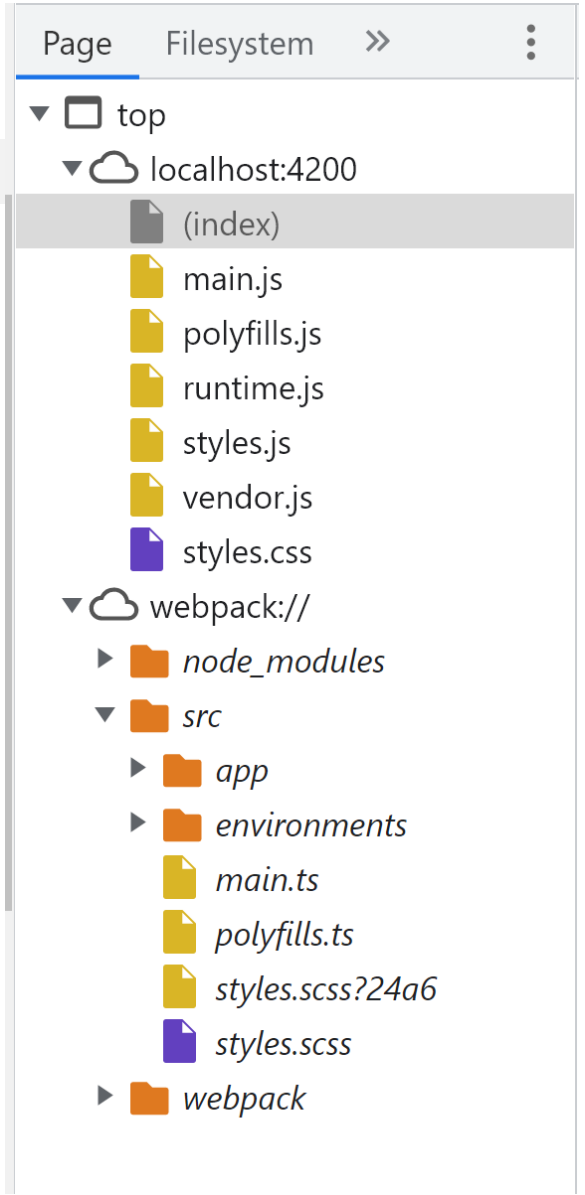


- And in the DOM inspector, see what our template has be transformed in:

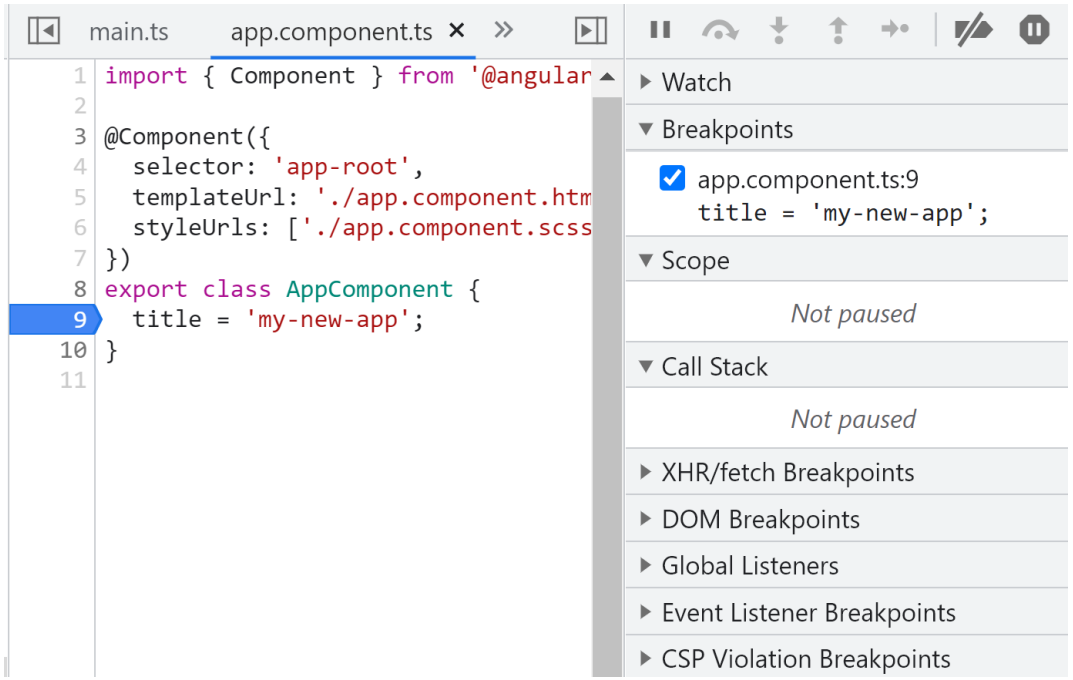


Debugging

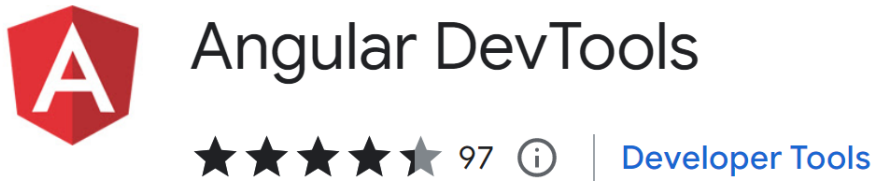
Similarly, we can see the **sources** tab, with all the details:



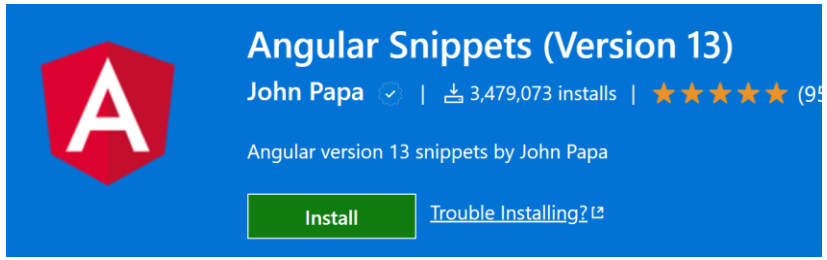
We can even put **breakpoints** in TypeScript code.



Some additional productivity tools



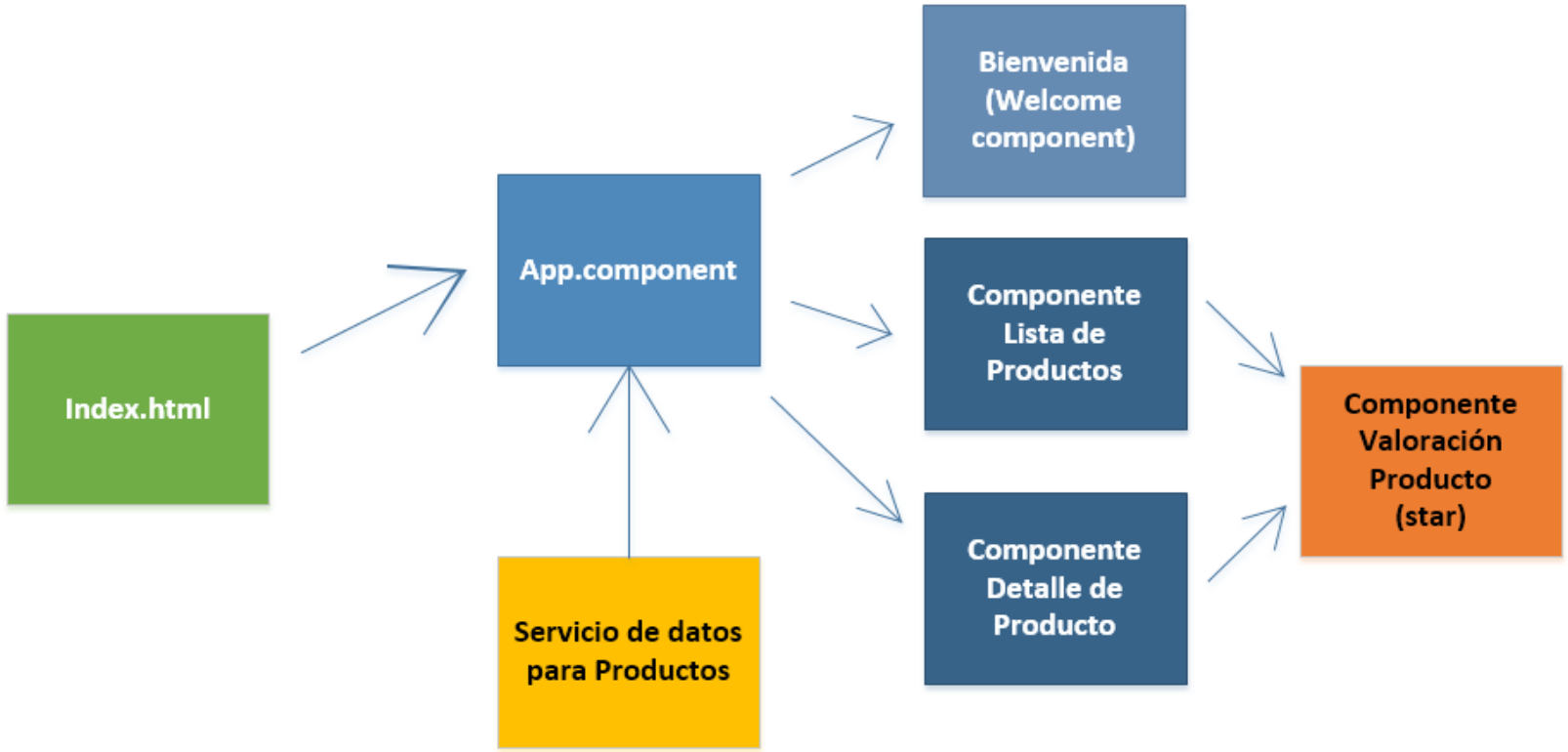
Angular DevTools
https://chrome.google.com/webstore/detail/angular-devtools/ienfalfjdbdpebioblfackkekamfmbnh



Angular TypeScript Snippets for VS Code
https://marketplace.visualstudio.com/items?itemName=johnpapa.Angular2

The project

We are going to propose a simple product management application, and we are going to see the different aspects of Angular through its construction.



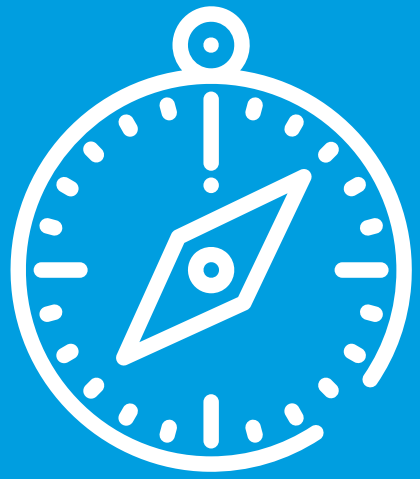
OBJECTIVE

Let's put it into practice: Tasks/Projects App

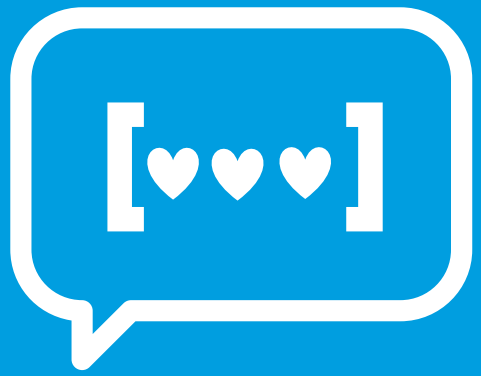
INSTRUCTIONS

1. Create the skeleton of an app that will be called TasksProjectsApp

Crea el esquel



Next steps



We would like to know your opinion!

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

netmind
a bts company

Thanks!

Follow us: