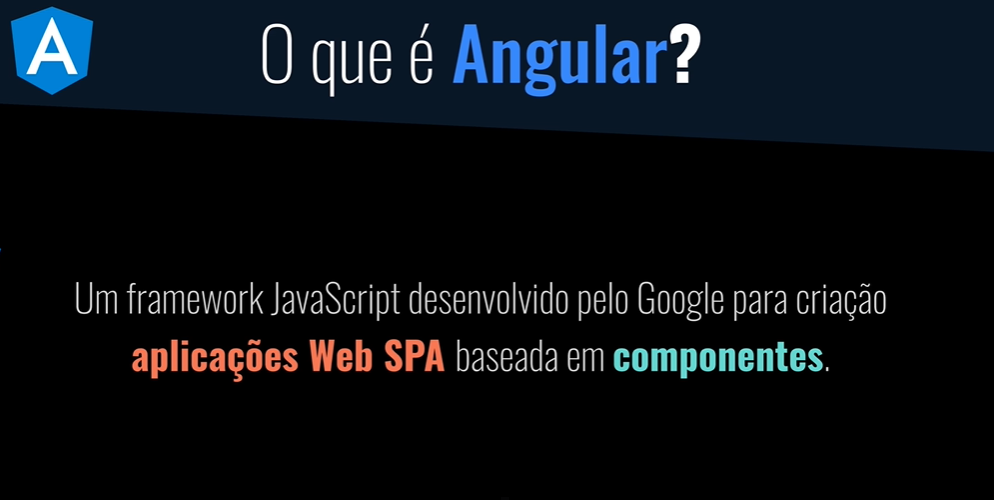
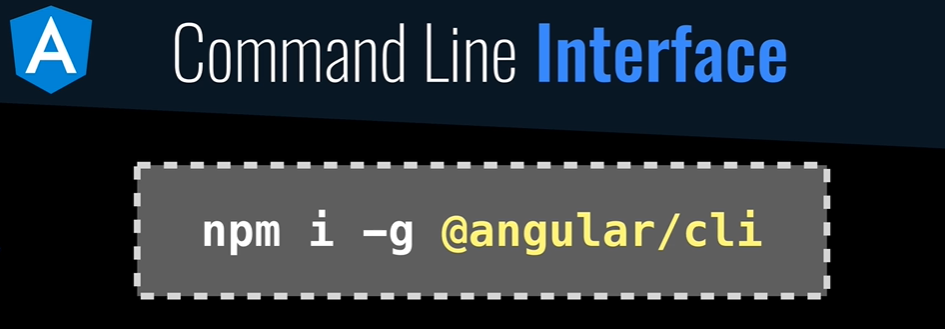
**Angular 9**





Com o Angular usamos também muito o CLI (Command Line Interface). Para instalar o CLI do Angular, segue abaixo:

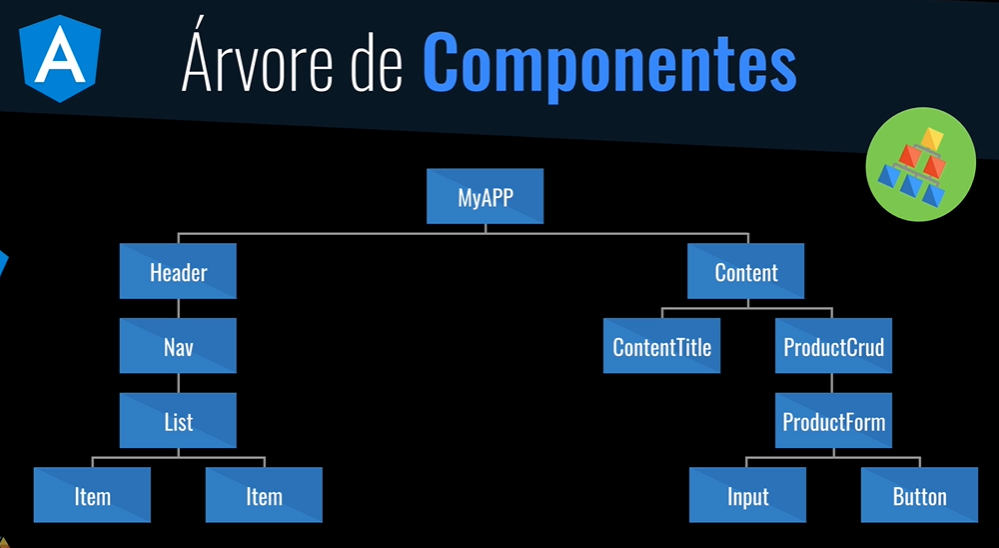


Após instalar o CLI do Angular através do terminal, podemos criar uma nova aplicação usando o Angular através do comando abaixo. Lembrando que “ng” vem de “Angular” e “minha-app” é o nome da aplicação que queremos dar:

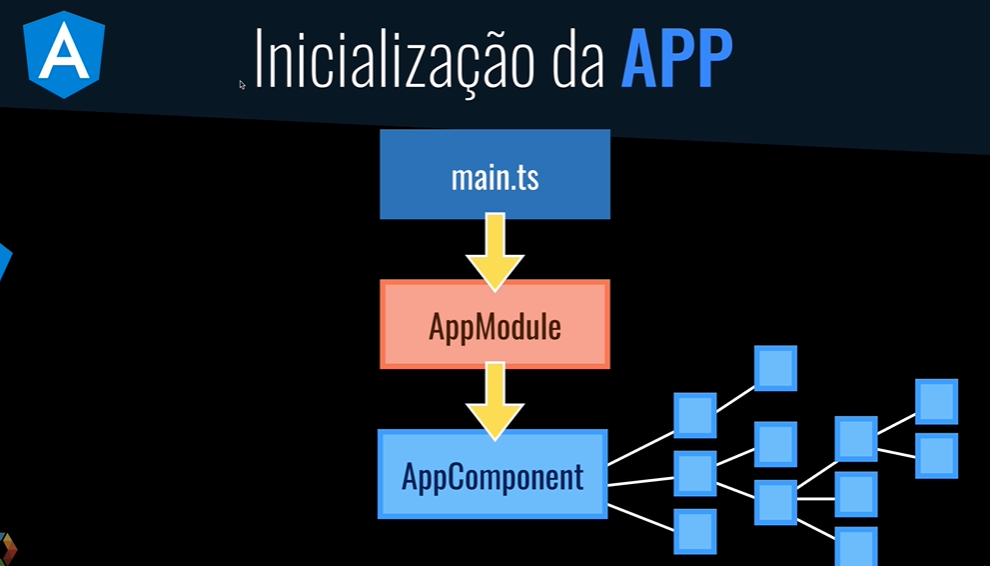




No Angular temos o uso de componentes e como todos os frameworks que usamos (seja React, Vue, etc) utilizamos uma árvore de componentes:



Assim que formos criar um app, criamos o arquivo main.ts (em typescript). **Após nossa aplicação chamará um AppModule, que é o módulo que contém o componente principal da nossa aplicação. Depois esse AppModule pode chamar AppComponent, que é o COMPONENTE que contém e dá acesso a toda a árvore de componentes da nossa aplicação**.

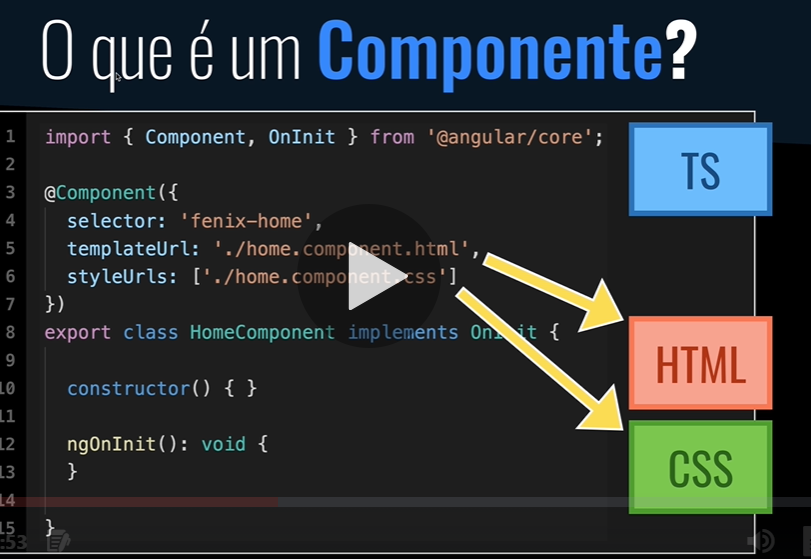




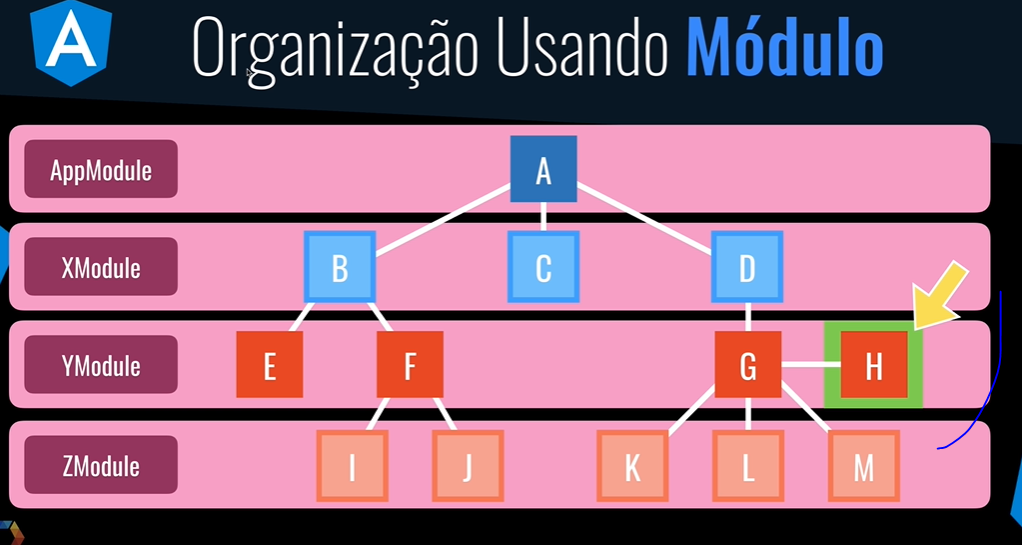
Quando criamos um componente em Angular contendo seus comportamentos (TS), estilos (CSS) e estrutura (HTML), é gerada uma tag padronizada por nós, nesse caso abaixo, utilizei uma tag para representar meu componente de home do meu app:



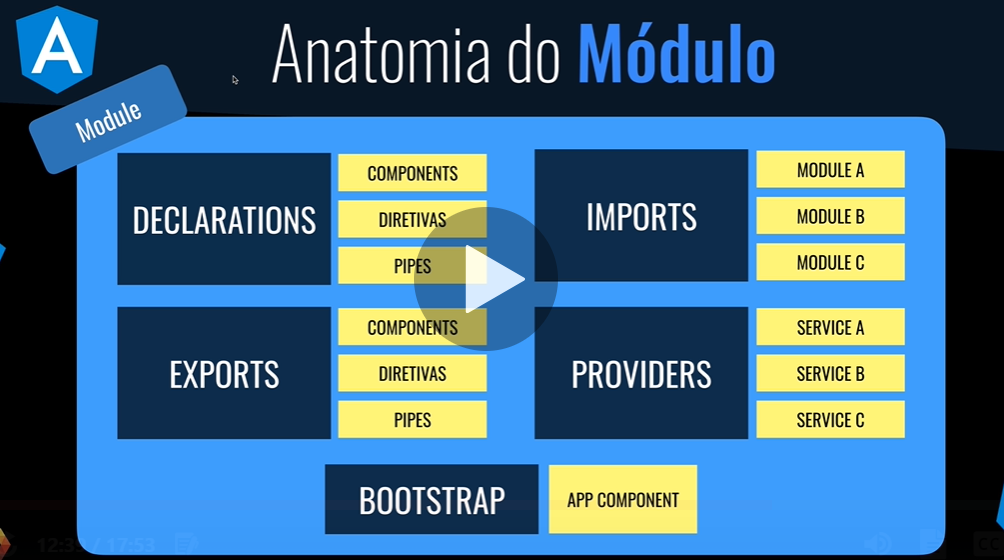
Através do arquivo TS conseguimos definir o nome da tag do componente e encontrar onde está o estilo (CSS) e estrutura (HTML) do meu componente para montar este componente:





**Com essa separação de componentes por módulos, podemos dizer se o componente de um módulo em questão estará disponível para uso de outros componentes de outros módulos de fora, ou este componente estará disponível para uso por outros componentes somente dentro do módulo em questão.** 

Dentro de cada módulo temos 5 atributos que são usados para configurar o módulo:



* **Declaration**: serve para declarar todos os componentes, diretivas, pipes que fazem parte do módulo;
* **Exports**: para tornar um componente visível para fora do módulo, devemos colocar no exports, que serão exportados, visíveis para fora do módulo;
* **Imports**: serve para importar outros módulos, um módulo pode depender de outro módulo. Lembrando que se eu importar no módulo B o módulo A, só vou conseguir usar os componentes do módulo A que estarão como exports no módulo A;
* Providers: aqui vamos declarar os nossos services dos módulos.
* Bootstrap: **que é o componente que será carregado no módulo. Ele é só preciso no AppModule, que é o Módulo principal da nossa aplicação e serve para carregar o componente principal da nossa aplicação**.



**Podemos criar componentes, módulos, projetos Angular através do CLI do Angular no terminal.**

**O arquivo angular.json é basicamente o arquivo que teremos algumas configurações do projeto.**

Para conhecer um pouco mais dos arquivos do projeto Angular, segue abaixo:

* Na pasta app, possuímos o arquivo de **AppModule** **principal** e componente do nosso app.
  + Estrutura do componente APP contendo os estilos, estrutura (HTML), seletor da tag HTML a ser usada:

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

@Component({

  selector: 'app-root',

  template: `

    <!--The content below is only a placeholder and can be replaced.-->

    <div style="text-align:center" class="content">

      <h1>

        Welcome to {{title}}!

      </h1>

      <span style="display: block">{{ title }} app is running!</span>

      <img width="300" alt="Angular Logo" src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4ybDE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xTDY2LjggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzTDEyNSA1Mi4xem0xNyA4My4zaC0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg==">

    </div>

    <h2>Here are some links to help you start: </h2>

    <ul>

      <li>

        <h2><a target="\_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>

      </li>

      <li>

        <h2><a target="\_blank" rel="noopener" href="https://angular.io/cli">CLI Documentation</a></h2>

      </li>

      <li>

        <h2><a target="\_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>

      </li>

    </ul>

    <router-outlet></router-outlet>

  `,

  styles: []

})

export class AppComponent {

  title = 'frontend';

}

* + Estrutura do ModuleAPP (usando @NgModule) contendo as declarations (componentes do módulo), imports (outros Modulos que estão sendo importados), providers e bootstrap (que define o componente principal a ser executado):

import { NgModule } from '@angular/core';

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

import { AppRoutingModule } from './app-routing.module';

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

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

* Na pasta assets, colocamos fontes, etc que podemos usar no nosso projeto;
* Na pasta enviroments temos a definição se é é produção ou em teste o nosso projeto;
* Ainda temos dentro da pasta src o index.html, que é a página html principal do projeto (contendo a tag do componente que quero mostrar no body da página):

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Frontend</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="icon" type="image/x-icon" href="favicon.ico">

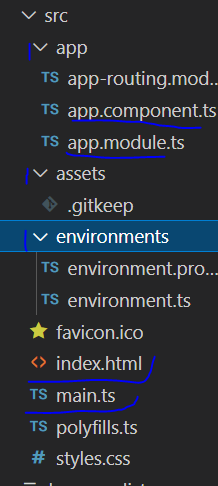
</head>

<body>

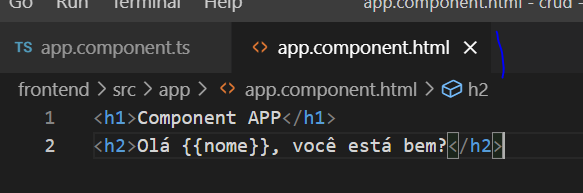
  <app-root></app-root>

</body>

</html>



Mas como você percebeu, no meu componente App, o template HTML está inline, vou criar dentro da pasta “app” um html para ser como template e, dentro do componente App, aí eu importo através da URL:



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

@Component({

  selector: 'app-root',

  templateUrl: 'app.component.html'

})

export class AppComponent {

  nome = 'Maria';

}

Tem algo legal também é que podemos exportar uma variável na parte de exportação do componente (em azul acima) e depois utilizar no HTML do meu componente utilizando o duplo mustache {{ }}, conforme abaixo:

<h1>Component APP</h1>

<h2>Olá {{nome}}, você está bem?</h2>



**OBS:**

**No projetinho que estamos fazendo para utilizar o Angular, temos um backend que tem um servidor local com uma base em JSON de produtos. Utilizamos o framework JSON-server para fazer este servidor local ficar assistindo um JSON para nos servir de base JSON.**

**E estamos usando o Angular em outro terminal para a parte do frontend.**

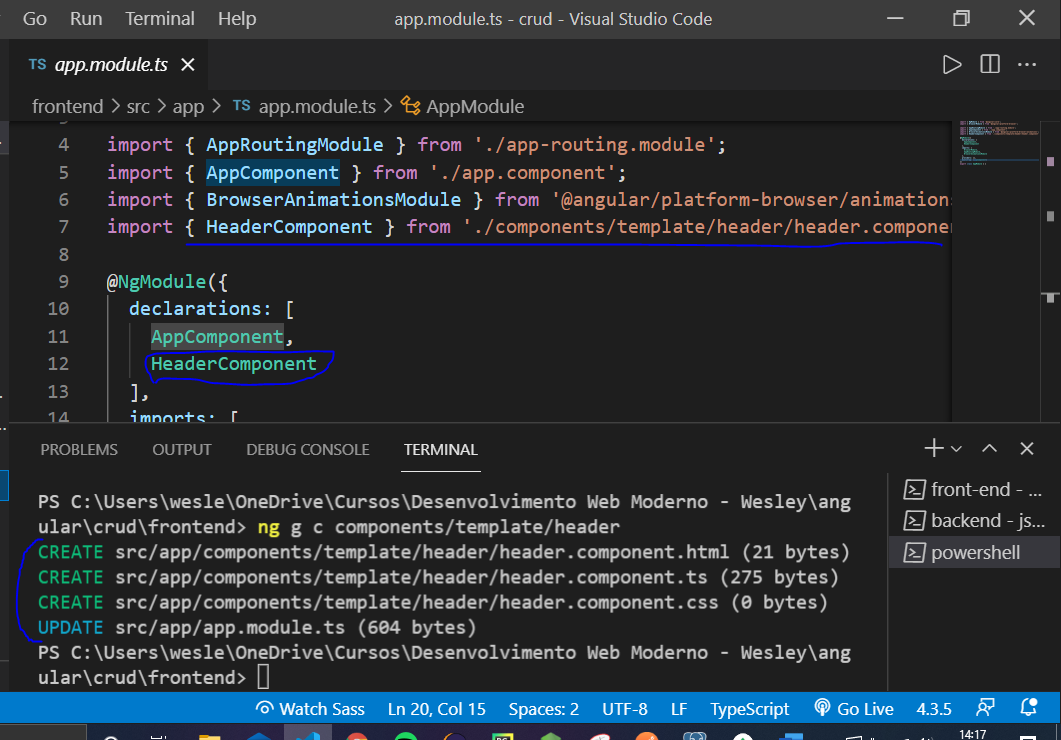
**Basicamente nosso projeto se trata de uma SPA (Single Page Application) que trará dados do servidor para realizar procedimentos de CRUD (Create, Read, Update, Delete) dados.**

Alguns comando no CLI para realizar ações com o Angular:

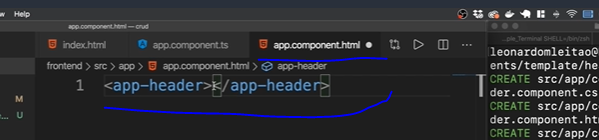
* **ng generate component** ou **ng g c**: para gerar um componente. E aí passamos também o caminho para criação deste nosso componente. Ex: **ng g c** **componentes/template/header,**

**onde em amarelo é o comando para criar o componente e em azul a pasta onde salvará o componente.**

* **Lembrando que ao criar o componente desta forma, automaticamente o Angular já cria no caminho deste componente os arquivos de estilo, html e typescript do componente, além de já registra-lo no AppModule como um componente (declarations).**

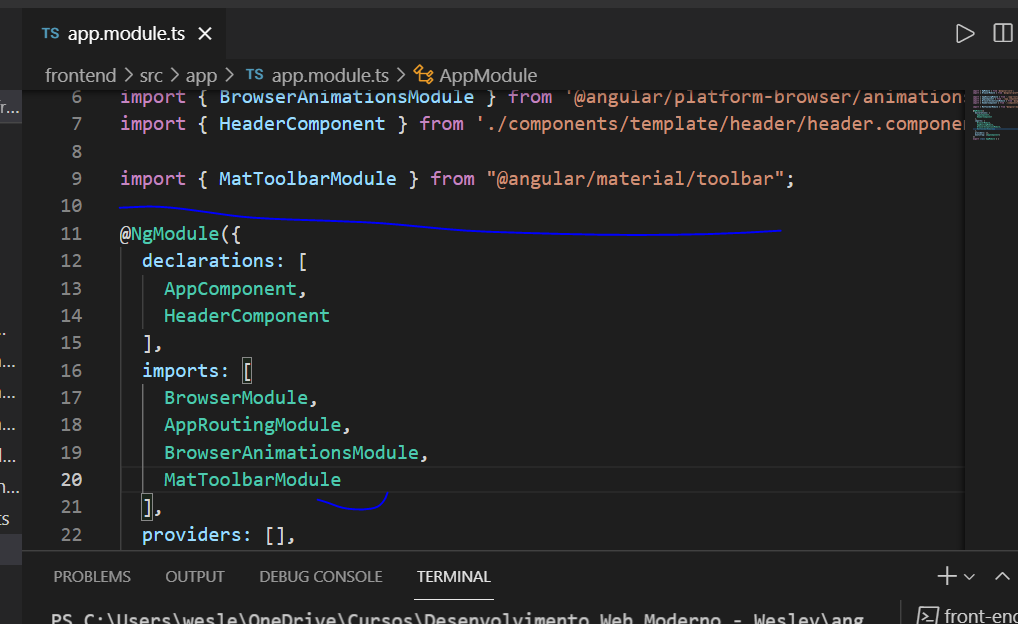


Agora que já temos esse componente de header criado, eu posso, dentro do HTML do componente principal do App referenciar o componente do header.



**Então, só lembrando, que se quisermos usar outros componentes ou módulos para dentro do nosso App, dentro de AppModule devemos importar esses outros componentes ou módulos.**

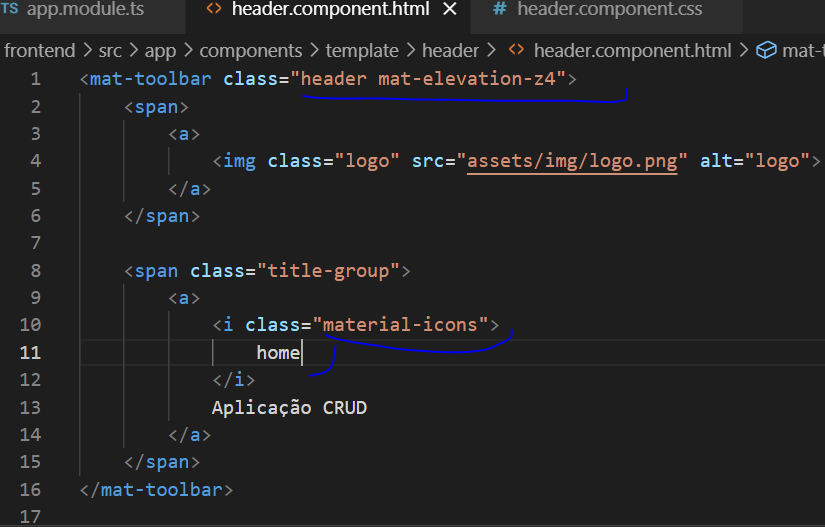
**No caso abaixo, eu precisava de um módulo de Toolbar para colocar no meu Header. Para isso, fui lá no Módulo de Toolbar do Angular e importei ele dentro do meu AppModule, para poder usar no meu header. Importei e depois coloquei dentro da especificação de import do AppModulo para dizer que estou importando este módulo para utilizar no meu App:**



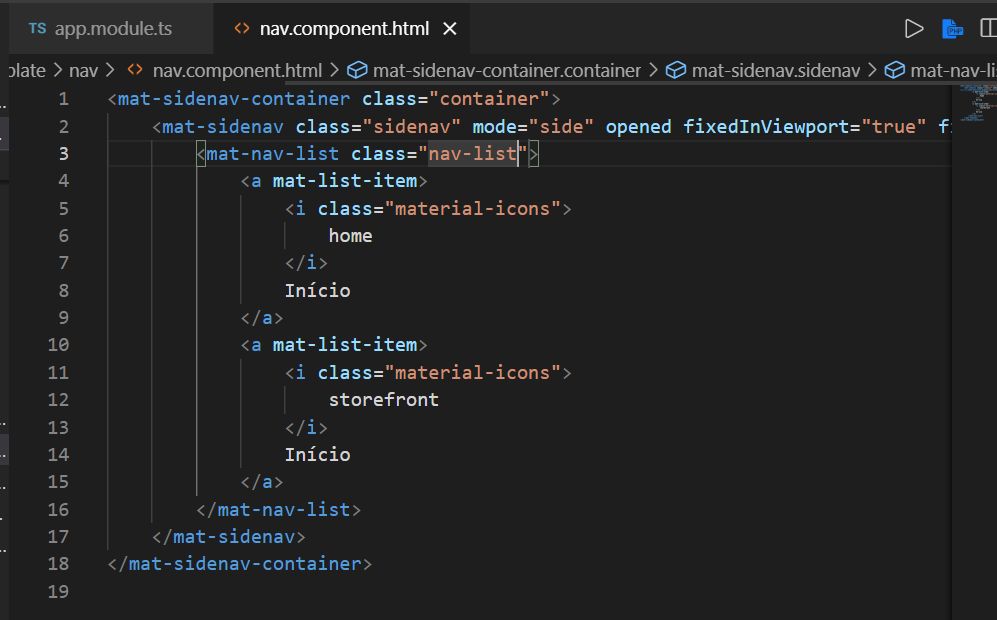
**Agora posso usar no HTML do meu componente Header o componente de Toolbar do Angular:**

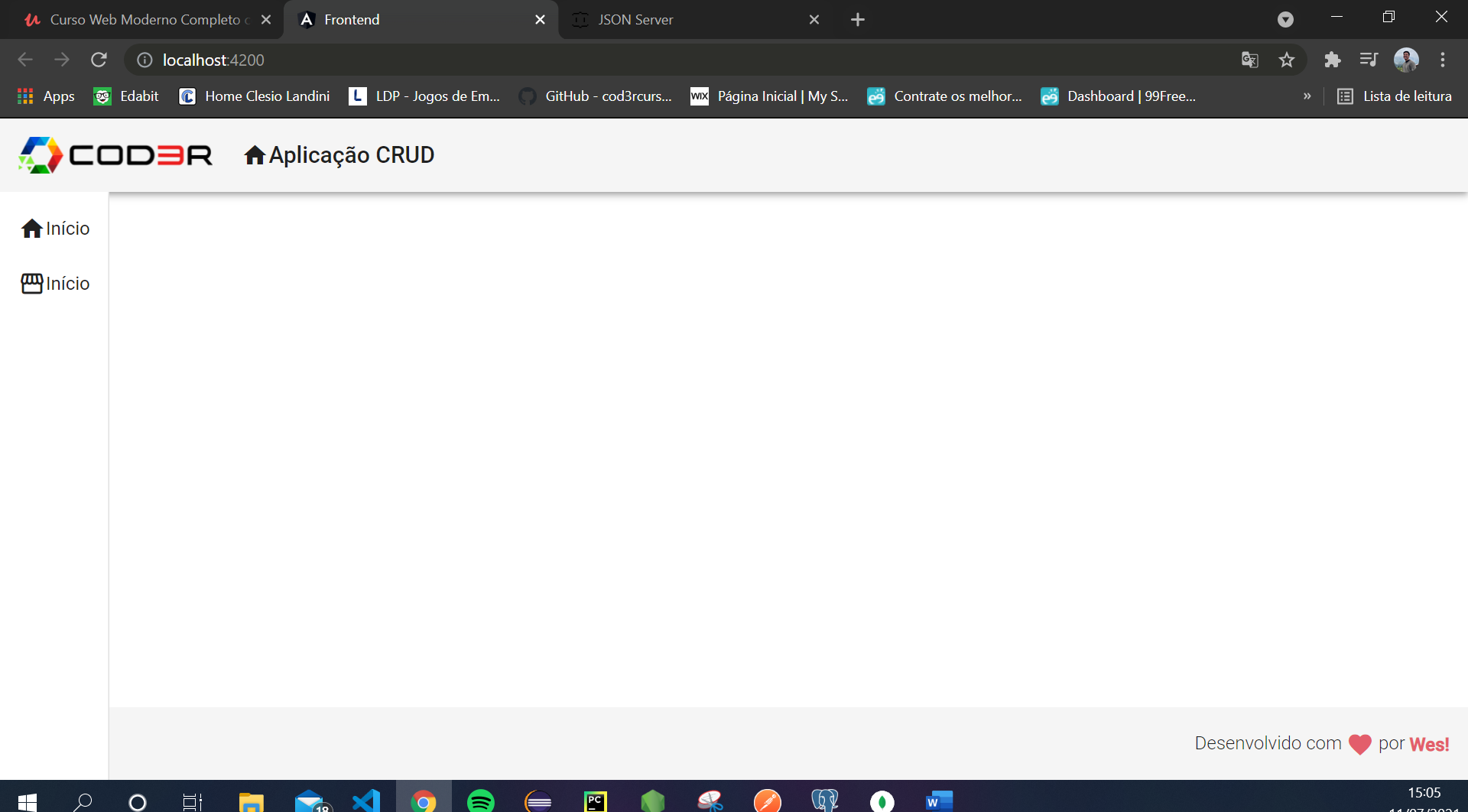


**A gente pode usar a biblioteca Material (a mesma que usei para colocar o Toolbar) para estilos nos nossos componentes. Essa biblioteca Material já é do Angular (como se fosse o Bootstrap para estilos do Angular) e nós podemos importar ela dentro do AppModule (como fiz acima para usar no header) para utiliza-la. No exemplo abaixo eu utilizei para colocar um efeito de sombra no toolbar e também para colocar ícones no meu menuzinho.**



**No exemplo abaixo estou usando o Materials para criar uma SideNav com NavList:**





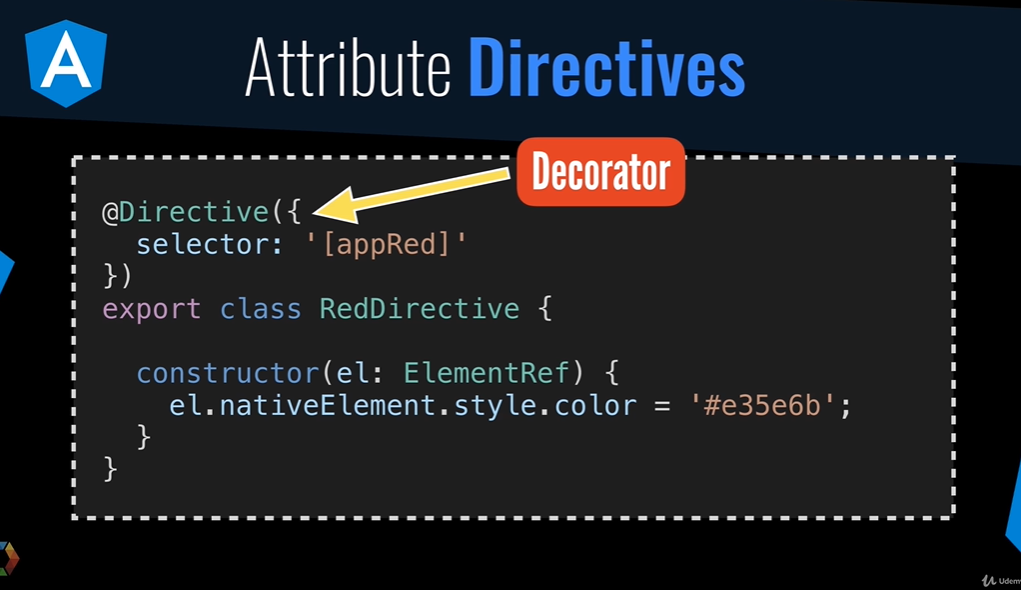
**Antes de prosseguirmos vamos ver agora um série de elementos do Angular:**

* **DIRETIVAS:**

**Tem dois tipos de diretivas: as diretivas de Atributo e as diretivas Estruturais.**

* + **Diretivas de atributo:**

Alteram a aparência e o comportamento de um elemento, componente ou outra diretiva. Ela foca em alterar a aparência ou comportamento. A parte de aparência se diz sobre a parte de CSS e a parte de comportamento quer dizer que coloco uma diretiva em um determinado botão e esta diretiva faz com que tenha uma chamada a uma ação no backend, por exemplo.



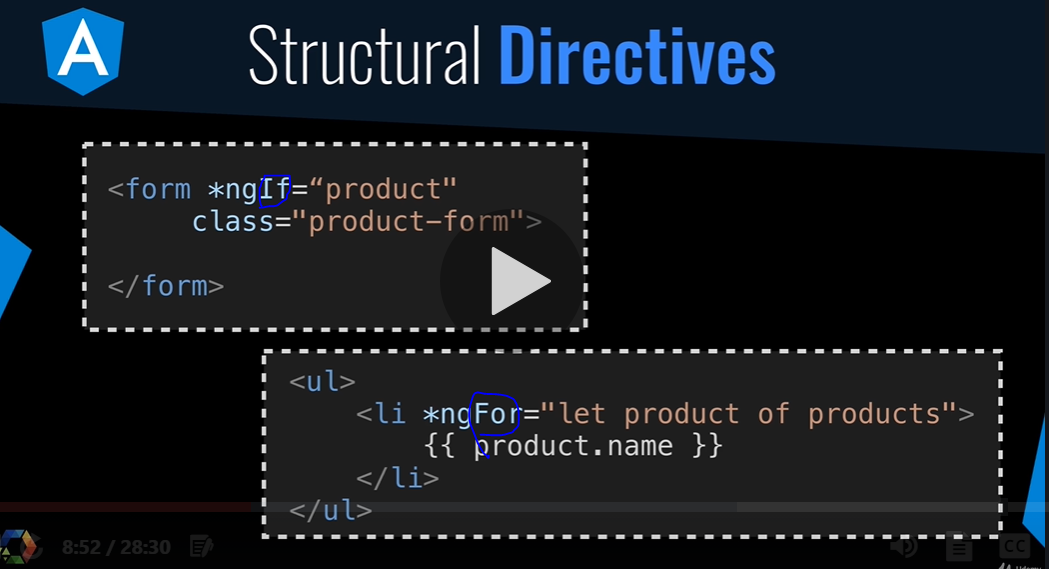
No exemplo acima criei uma diretiva para alterar a cor de um elemento e também passei o selector dele para passar a um elemento HTML, conforme uso abaixo:



Ou seja, usei a diretiva de atributo que criei para aplicar uma cor diferente ao meu elemento HTML.

* + **Diretivas estruturais:**

Altera o layout adicionando ou removendo elementos da DOM. Ou seja, adiciona ou remove elementos da DOM. **Para diferenciar uma diretiva de atributo para uma diretiva estrutural é que a diretiva estrutural tem um “\*ng” na frente.**



Na primeira diretiva estrutural do exemplo, vai exibir ou não o formulário se o produto estiver definido, estiver setado (estou utilizando o \*ngIf).

No segundo exemplo, eu acrescento novos elementos, na UL que é uma lista e aí faz um laço for sobre os produtos, adicionará elementos na DOM conforme forem existindo na lista de produtos. (estou utilizando o \*ngFor)

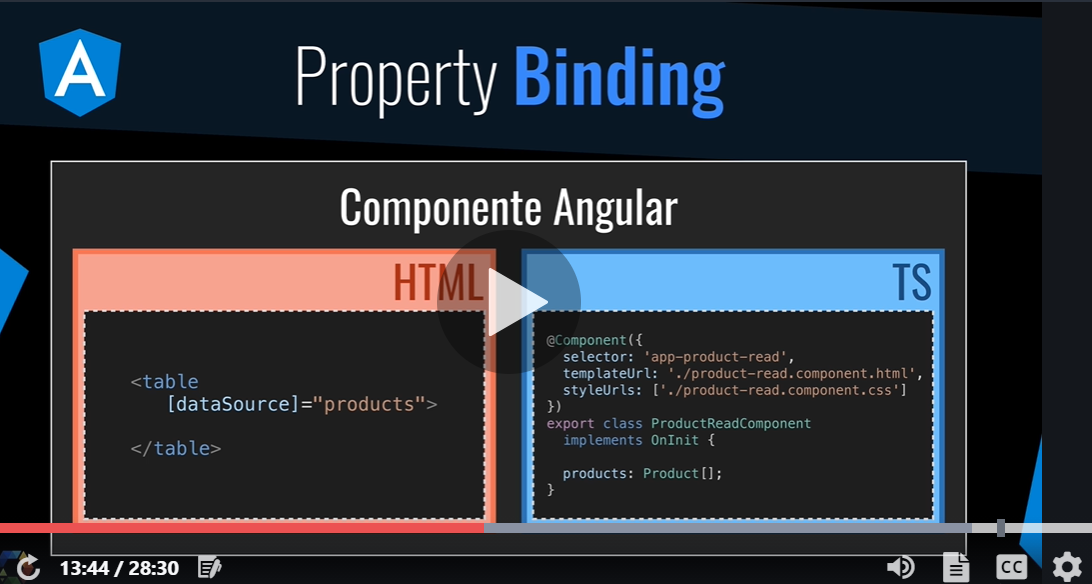
Neste caso, estas duas diretivas já vem no angular prontas para utilizarmos.

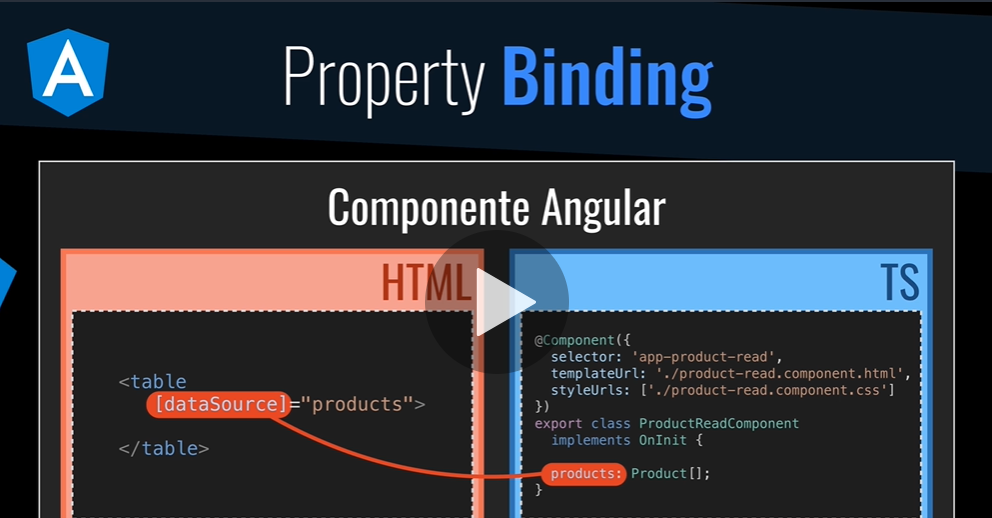
* **PROPERTY BIND:**

O BINDING serve para que tenhamos uma comunicação entre o TypeScript do meu componente com o Template HTML do meu componente. Um exemplo, lá no meu TypeScripts do meu componente eu estou exportando para ser utilizado um array de produtos, do tipo Products. Para que eu possa utilizar este array no template HTML eu uso o bind, colocando no template HTML um atributo [dataSource] com o o valor sendo igual o nome do array que quero utilizar. Com isso, conforme exemplo abaixo, eu consigo montar uma tabela de produtos HTML utilizando o bind pegando o array de produtos que exportei no arquivo TypeScript, fazendo a comunicação entre o TypeScript com o HTML. Binding quer dizer ligação em inglês.

**Ou seja, com o Binding podemos fazer a ligação de uma variável no TS com o template HTML.**

Veja exemplo abaixo:





* **EVENT BIND:**

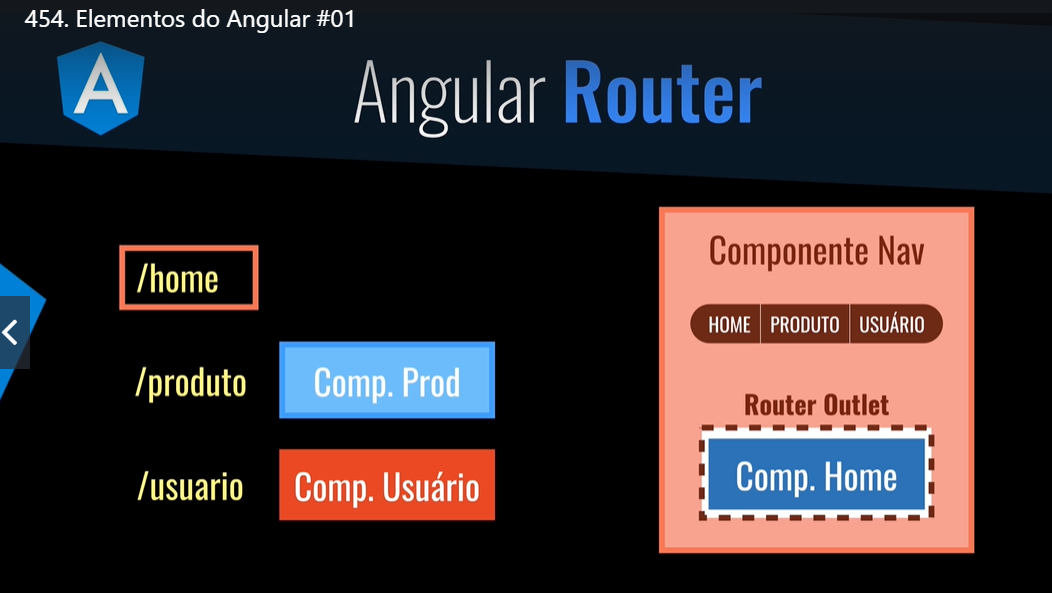
Serve para ligar um evento no template HTML com um método do TypeScript. Ou seja, por exemplo: chamar um método do TS quando eu realizar um evento de click:



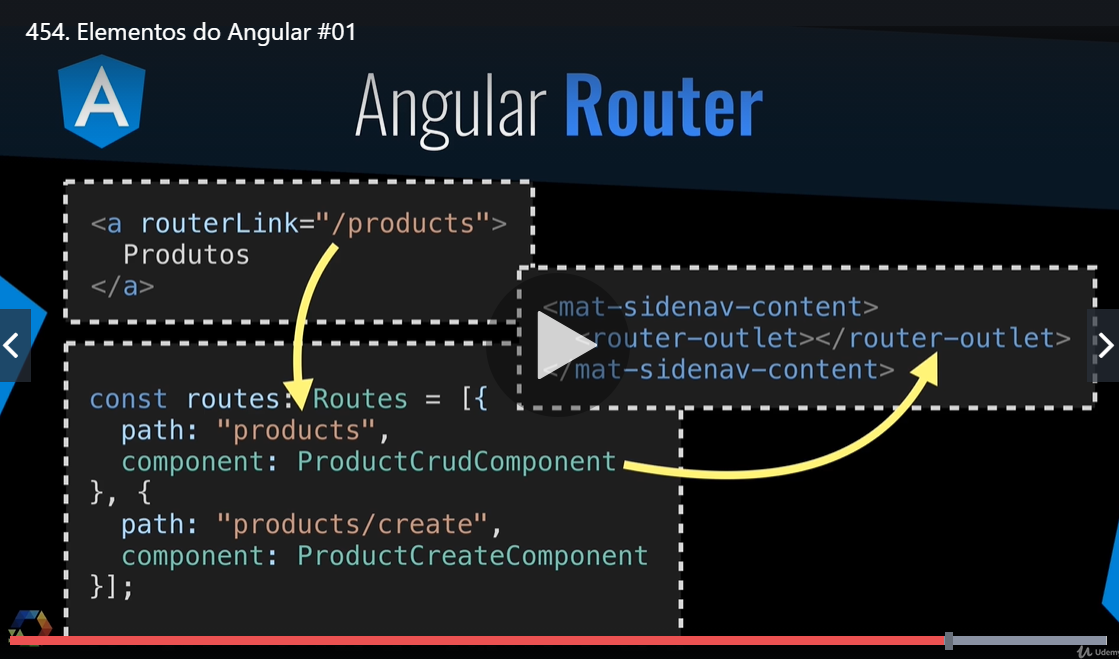
**Para usar o Two-Way Data Binding, que é quando eu faço a ligação de um elemento HTML com uma variável do TS e quando há a atualização desta variável no TS ou atualização no elemento HTML, haja também a atualização do outro lado(ex: atualizo a variável no HTML e automaticamente atualiza no TS e atualiza no TS a variável e no HTML também atualiza), usamos a sintaxe abaixo, conforme exemplo:**



* **ROUTER:**

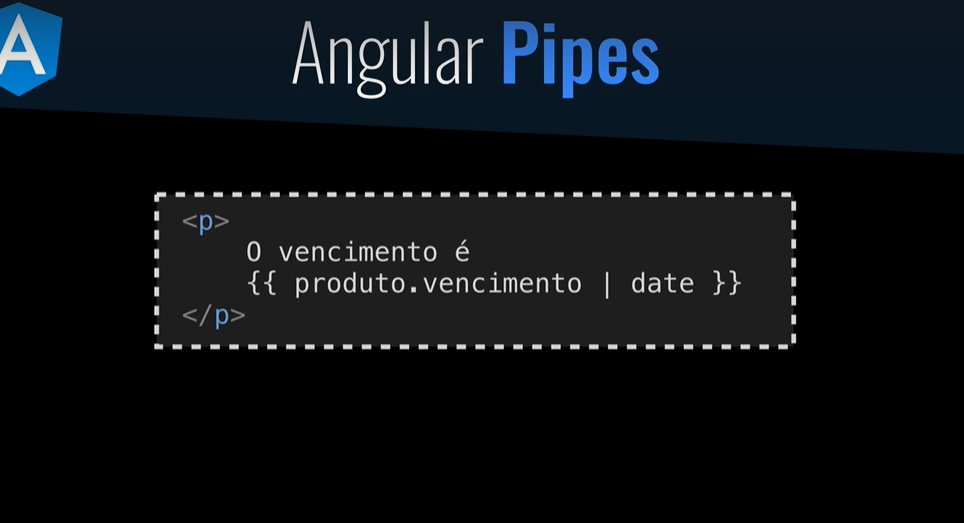
Nas nossas aplicações temos sempre a necessidade de quando o usuário mudar a URL termos que trocar o componente que está sendo exibido na tela. Ou seja, quando o usuário vai para a URL “/home”, tem que aparecer o componente de Home, quando o usuário vai para a URL “/produto”, tem que aparecer o componente de Produto e por aí vai. Para termos o mapeamento destas rotas para aparecer um determinado componente quando o usuário for para uma determinada URL, usamos o Router. 

Ou seja, após mapearmos os componentes que devem aparecer após o usuário for para uma URL, terá uma parte da aplicação que se chama Router Outlet, que é o lugar que será carregado o componente de acordo com a URL que o usuário selecionar.



* **Pipes:**

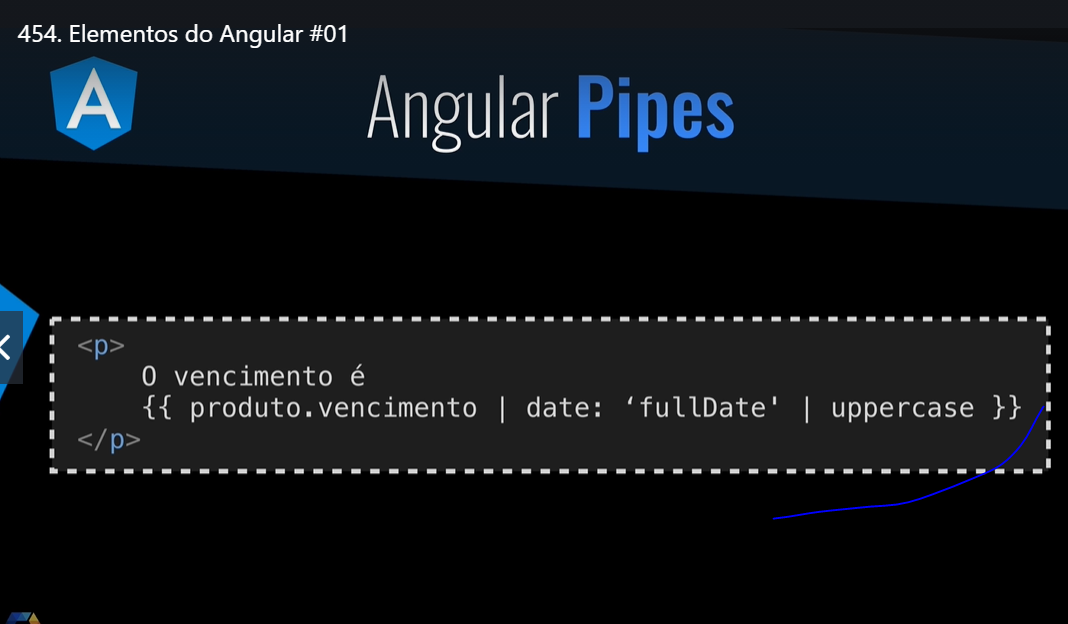
Pipe é a formatação de um dado que fazemos nos nossos componentes.



Ou seja, eu to usando uma variável do meu componente no HTML e utilizei o pipe para dizer que quero que esta variável seja retornada como um padrão de data, por exemplo.



Podemos também ter uma cadeia de processamentos do dado com o Pipe:



* **Observables:**

O Angular utiliza por baixo dos panos um conceito de programação reativa com um framework chamado ReactiveX (rxjs), que também tem para outras linguagens. Uma programação reativa é, ou seja, um código reativo é um código só será chamado de forma reativa, sendo assim, precisa acontecer algo externo ou algum evento para daí o código ser executado. E a gente usa isso dentro do Angular. Para isso, importamos, como abaixo, no nosso projeto em angular o Observable do framework rxjs para usarmos este conceito de programação reativa.

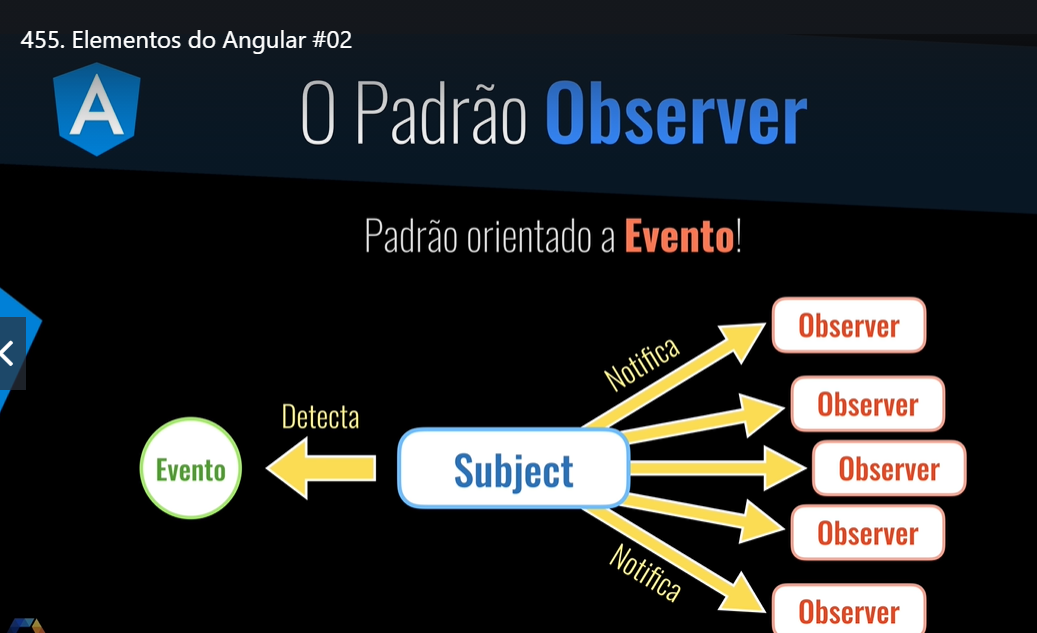


O padrão de projetos e códigos Observer é um padrão orientado a evento.

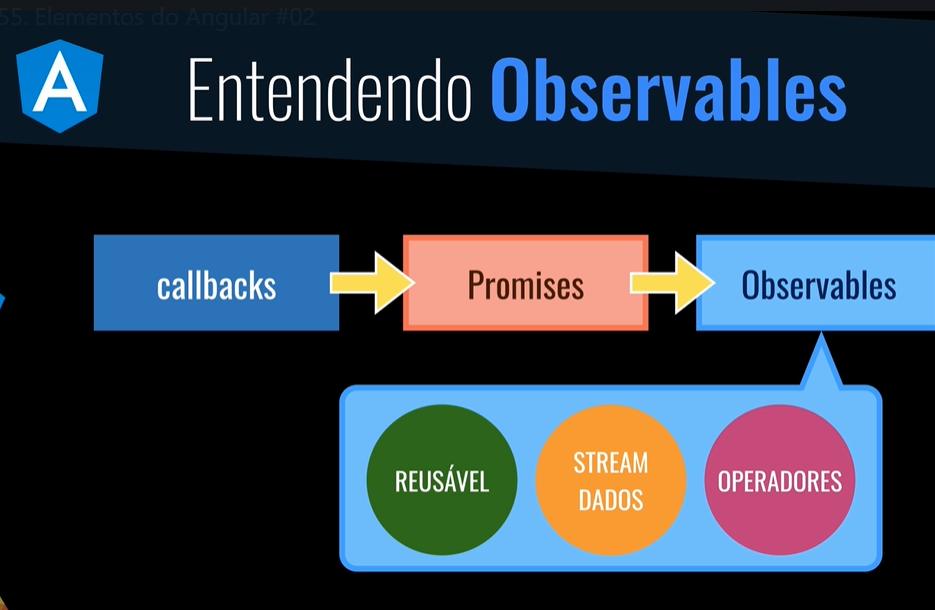
Neste padrão existe um cara chamado Subject, que é o cara que tem a capacidade de monitorar quando o Evento acontece.

Também neste padrão tem os Observers que são os códigos ou as pessoas que estão interessadas em um determinado evento. Mas neste projeto, como temos o Subject que consegue monitorar os eventos, podemos fazer com que os Observers deleguem para os Subjects esta monitoração dos eventos e quando um determinado evento acontecer, o Subject possa alertar os Observers que determinado evento ocorreu.

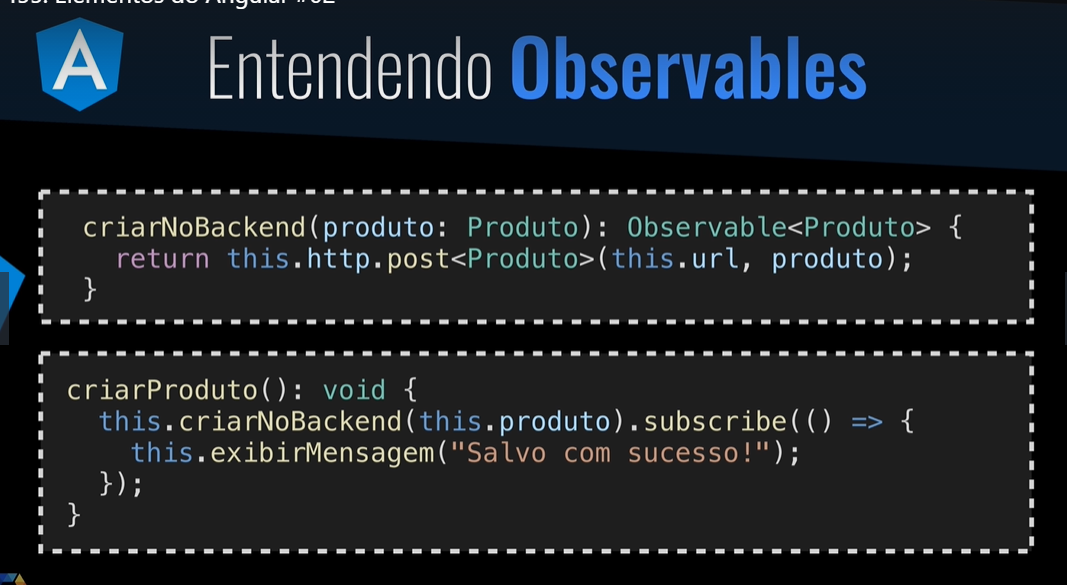
Cada Observer deve registrar para o Subject o desejo de receber alertas quando um evento acontecer para que o Subject possa realizar a monitoração e alertar quando um determinado evento acontecer.



Imagina que temos um evento de Compra e temos um Observador que é para enviar um e-mail ao usuário, temos outro observador que é para dar baixa no estoque, o Subject que é quem está monitorando este evento de compra que vai notificar o observador de e-mail para que ele possa enviar um e-mail e vai notificar também o observador de estoque para dar baixa no estoque.



**Os Observables são a evolução do que temos no JS de call-back (um função chama outra função dentro dos parâmetros) e das Promises.**



No código acima, usando o Observable, vamos explicar.

No primeiro bloco temos uma função que recebe como parâmetro um produto do tipo Produto, que trás um retorno do tipo Observable<Produto> (aqui estamos dizendo que esta função está interessada em algum monitoramento, neste caso o retorno de sucesso do Post no banco de dados da requisição). Essa função simplesmente faz um Post do tipo produto, na URL que passamos com parâmetro e com o conteúdo do parâmetro “produto que passamos.

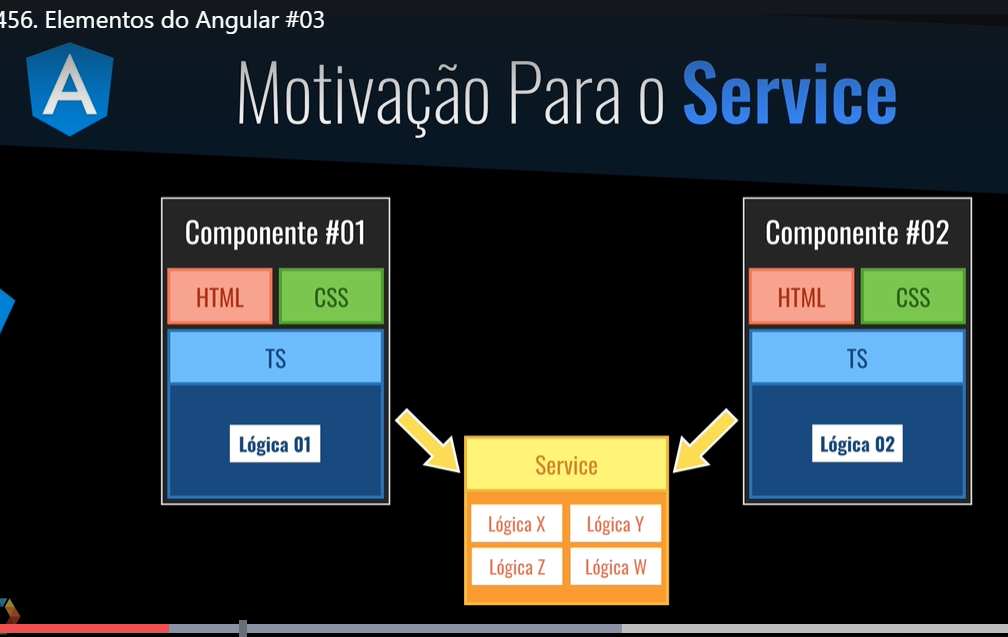
No segundo bloco criamos uma função que vai simplesmente servir para registrar (subscribe) e monitorar essa ação de post no banco de dados. Para isso passamos a nossa função com o subscribe e no parâmetro do subscribe passamos a função que queremos fazer quando der certo o POST, nesse caso mostrar a mensagem “Salvo com sucesso!”.

* **Services:**

São classes que tem como objetivo principal organizar e compartilhar métodos e dados entre componentes. Também posso usar services dentro de diretivas.

A ideia é que a parte visual sempre fique com os componentes e a parte não visual como é a organização do código e as responsabilidades a gente deixe com os Services.

Ou seja, os services servem para termos lógicas que podemos usar para várias componentes, não somente para um, e assim organizarmos quais componentes vão utilizar quais lógicas.



Teremos sempre services com lógicas coerentes de funcionalidades.

Para criarmos um Service podemos usar o comando abaixo:



Este Injectable significa que este Service pode ser usado e injetado nos outros componentes/classes, só que somente será instanciado uma vez, ou seja, mesmo podendo ser usado várias vezes, sempre será a mesma instância do Service (alias “root”).

Podemos também querer ter um Service com um instancia dedicada para um determinado componente, é o que chamamos de ElementInjector:

