



Desenvolvendo SPA com Angular

Otimizando o desenvolvimento de aplicações web com Single Page Application com foco em gerar uma melhor experiência ao usuário.

Agenda

- 1 Quem sou eu?
Um pouquinho sobre mim
- 2 Descontração
Vamos quebrar o gelo?
- 3 Introdução
Falando um pouco sobre o Angular
- 4 Desenvolvendo SPA com Angular
Vamos juntos criar uma Single Page Application
- 5 Dicas de conteúdo / Desafio
Bate papo / Explicação do Desafio

Quem sou eu?

- 10 anos na área de Tecnologia
- 3 anos de Avanade
- Graduada em Ciência da Computação
- Amo livros, filmes, séries e animes
- Comecei a pular corda na quarentena e amo cozinhar



Camila Ferreira Ribeiro

Full-Stack Developer
Senior Software Engineering





Acesse pelo site:

<https://kahoot.it/>

Ou baixe o app do

Kahoot!

Game PIN: **7013302**





Introdução ao Angular

Um breve resumo para se interar no tema

Sobre o Angular

Angular é um Framework que possibilita construir aplicações Web baseadas em HTML 5, CSS e JavaScript.

Ele permite organizar essas tecnologias e entregar uma aplicação que executa no Browser, capaz de consumir um ou vários serviços disponibilizados por um servidor.

Quando o Angular 2 foi anunciado, criou um pouco de confusão, pois o time de desenvolvedores, decidiu reescrever o Angular do zero. Isso quebrou a compatibilidade com a versão 1.

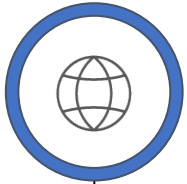


Os fatores que levaram a isso foram:

- 1) O Angular surgiu para apps simples
- 2) A API cresceu inconsistente
- 3) Performance ruim

A partir disso surgiu o Angular 2, que é um frame mais moderno, mais aderente aos padrões da web, e em muitos aspectos mais simples, com um padrão mais fácil de evoluir.

Fatores que
levaram
à mudança de
versão



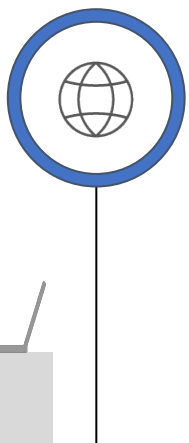
Os principais benefícios desse novo Angular são:

- 1) Mais aderente a padrões
- 2) Trabalha com a ideia de classes
- 3) Qualquer propriedade do DOM pode receber um valor dinamicamente sem a necessidade de criar diretivas.

Benefícios do
Angular 2



O que é SPA?



Single Page Application



No page refresh on request

Traditional Web Application



Whole page refresh on request





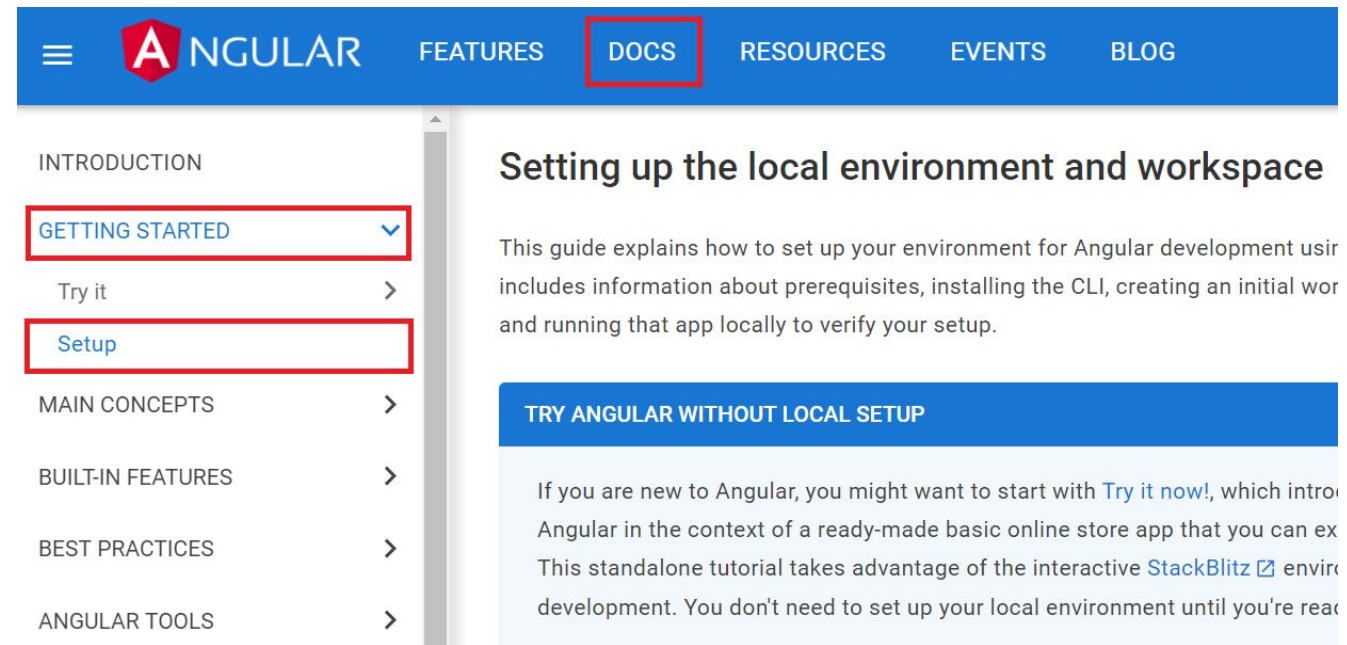
Desenvolvendo SPA com Angular

Construindo uma Single Page Application


Conhecendo os requisitos mínimos

No site: (<https://angular.io/>) é possível visualizar um conteúdo bem abrangente, onde encontramos: Tutoriais, Dicas, Ferramentas do Angular, Melhores Práticas, entre outros...

Na página (<https://angular.io/guide/setup-local>), podemos visualizar as instruções para configuração de seu ambiente, utilizando o Angular CLI. Seguindo este passo a passo, teremos nossa primeira aplicação.



Gerando nossa primeira aplicação

- Instalação do Node (<https://nodejs.org/en/>)
- **node -v** (versão instalada)
- **npm install -g @angular/cli** (necessário para realizar o build da nossa aplicação, criando componentes, diretivas, pipes)
- **ng v** (visualizando a versão do Angular que foi instalada)
- Escolha de uma IDE de sua preferência. Nesta aula irei utilizar o **Visual Studio Code**  **VS Code** (<https://code.visualstudio.com/>).



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [versão 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

C:\Users\c.ferreira.ribeiro>npm install -g @angular/cli
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
C:\Users\c.ferreira.ribeiro\AppData\Roaming\npm\ng -> C:\Users\c.ferreira.ribeiro\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

> @angular/cli@10.1.7 postinstall C:\Users\c.ferreira.ribeiro\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js

+ @angular/cli@10.1.7
added 57 packages from 51 contributors, removed 9 packages and updated 100 packages in 51.951s

C:\Users\c.ferreira.ribeiro>

C:\Windows\System32\cmd.exe
ou externo, um programa operável ou um arquivo em lotes.

C:\Users\c.ferreira.ribeiro\Documents\GitHub\SPA-Angular>ng v

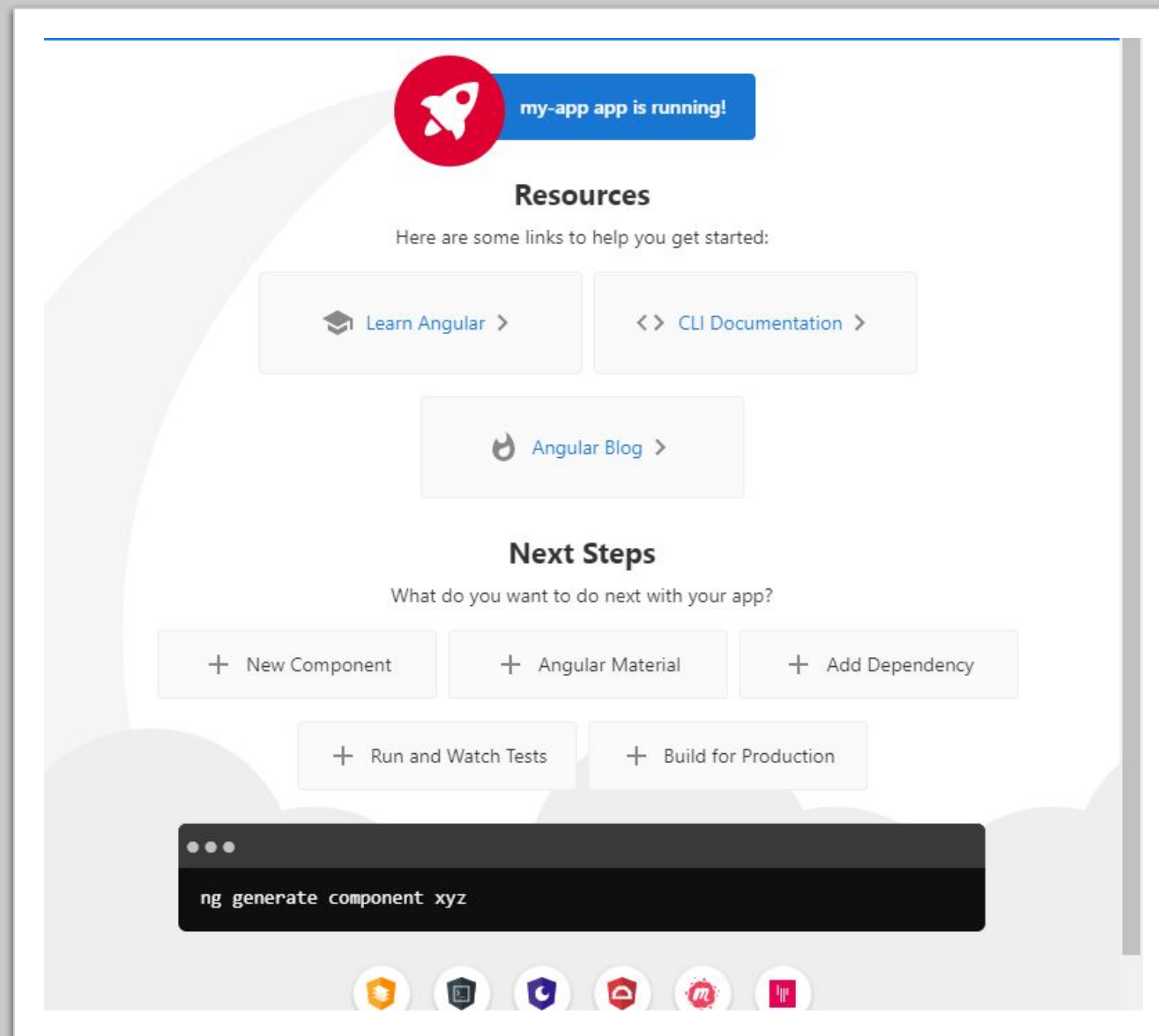
Angular CLI
Node: 10.16.0
OS: win32 x64

Angular:
...
Evy Workspace:

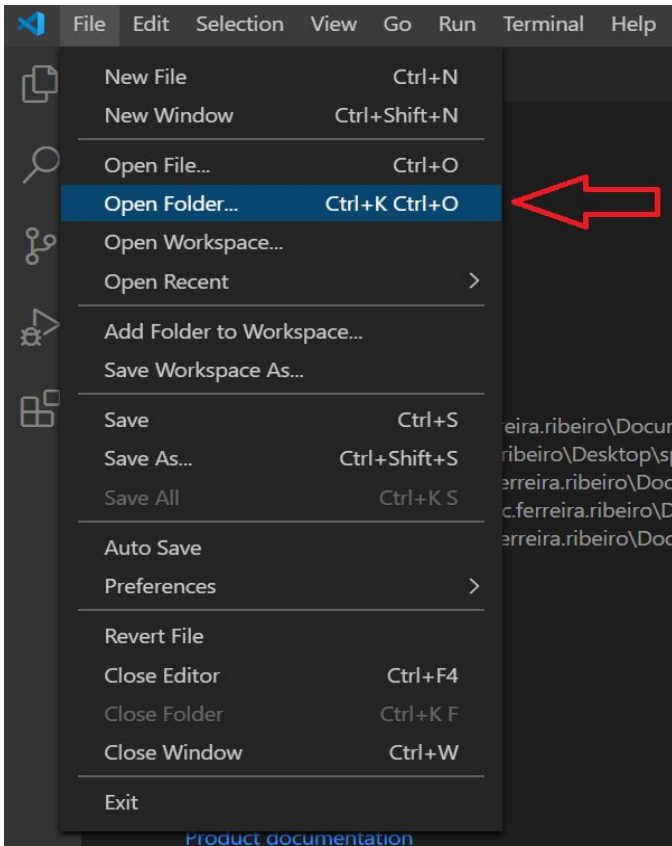
Package                                  Version
-----
@angular-devkit/architect               0.1001.7 (cli-only)
@angular-devkit/core                    10.1.7 (cli-only)
@angular-devkit/schematics              10.1.7 (cli-only)
@schematics/angular                    10.1.7 (cli-only)
@schematics/update                      0.1001.7 (cli-only)
```

- **ng new spaangular --prefix=spa**
- **Would you like to add Angular Routing?** Ao responder “Yes” ele cria um arquivo de rota em nossa aplicação.
- **Which stylesheet format would you like to use?** Neste caso, basta ir com a setinha e escolher CSS, que é a folha de estilo que iremos utilizar.
- **cd spaangular** (Comando para navegar até o seu projeto. Pode ser substituído abrindo o CMD diretamente na pasta da aplicação)
- **ng serve -o** (Comando para iniciar o servidor, e ficar observando os arquivos. A cada mudança nos arquivos, ele reconstrói “rebuild” a aplicação.)
- O comando **-o**, é uma abreviação para **--open**, e abre o seu browser padrão após finalizar a compilação)

Seu projeto inicial ficará parecido com este aqui:

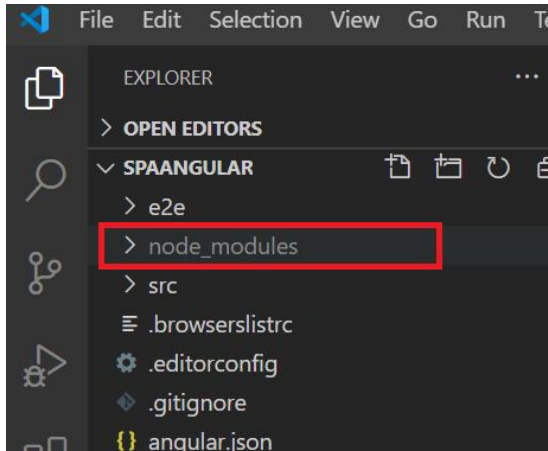


Abrindo seu projeto no Visual Studio Code

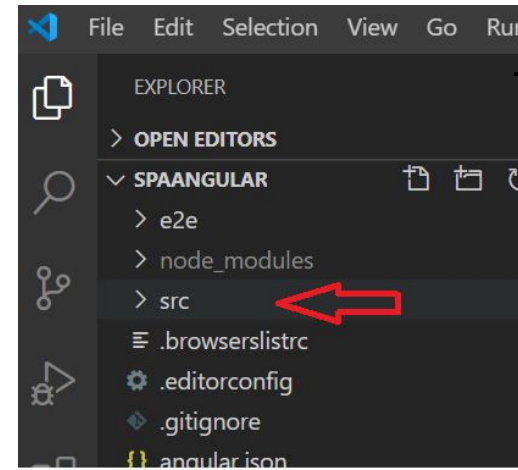


- Abrir a pastinha através do commando Ctrl+K / Ctrl+O.
- Este processo trará toda estrutura inicial do Angular para dentro de sua IDE.

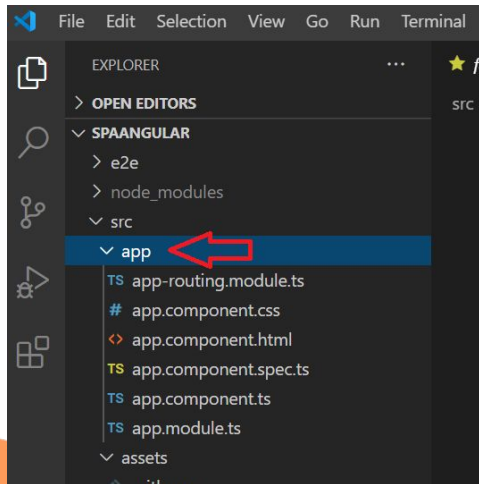
Conhecendo alguns diretórios



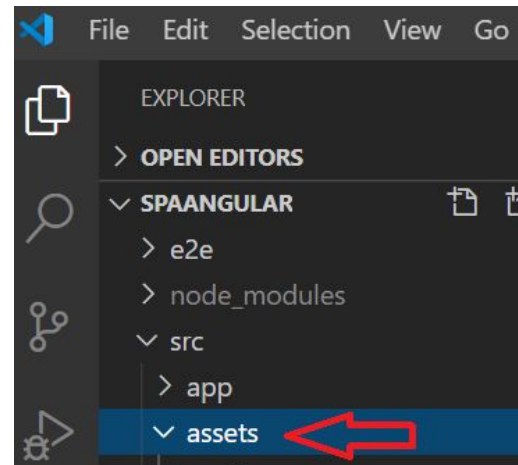
- Aqui estão todos os pacotes e dependências que o Angular NPM baixou e instalou na nossa aplicação.



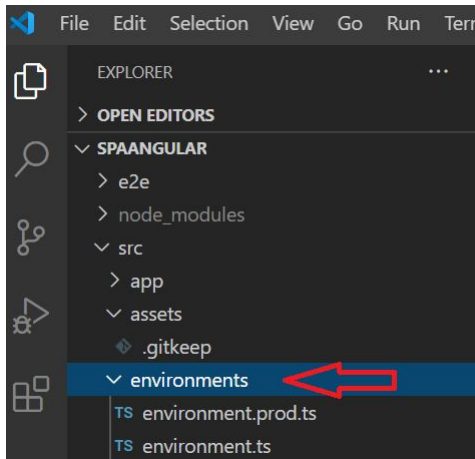
Dentro de SRC existem algumas pastinhas que ele já traz pronta e que fazem parte da estrutura inicial.



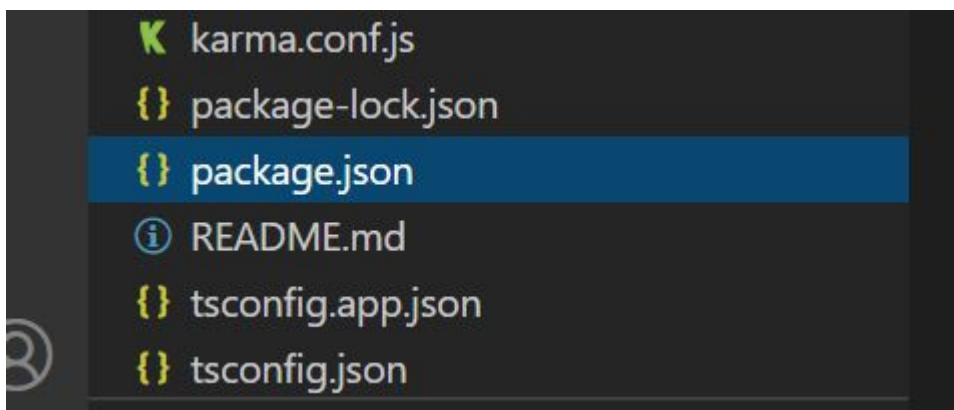
- Na pasta APP, ficam basicamente todos os nossos componentes e arquivos de modulo.



- Na pastinha Assets, colocaremos nossos arquivos de estilo comuns em toda a aplicação.



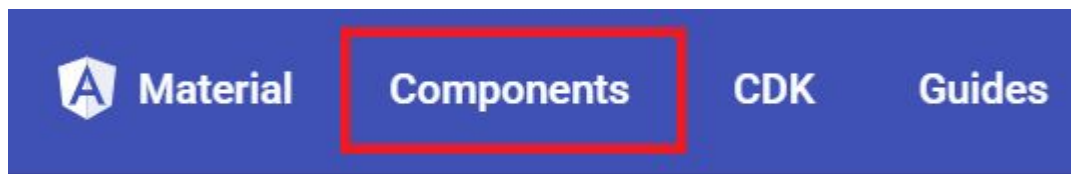
- Esta pastinha nos ajuda a criar diferentes ambientes para que a aplicação tenha um comportamento único em cada um deles, facilitando no deploy do código.



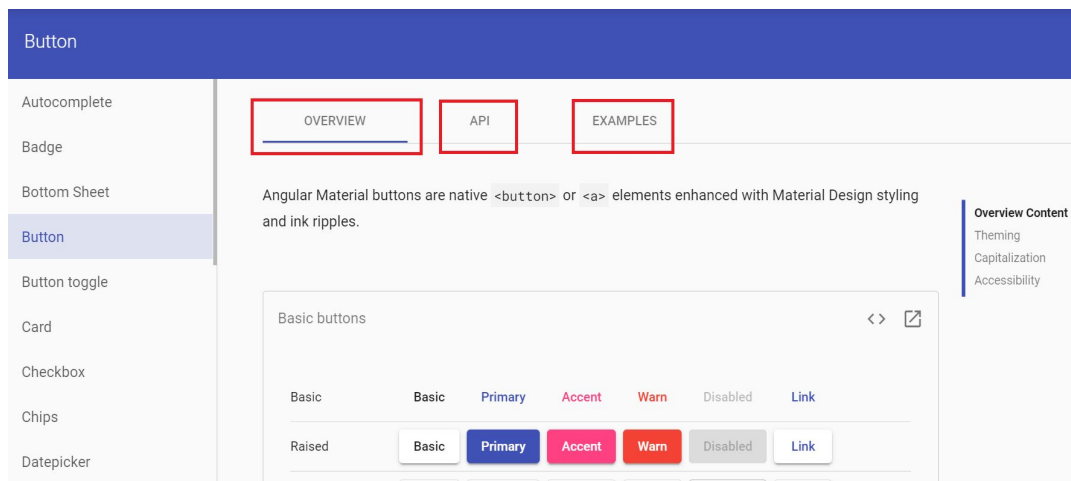
- Neste arquivo estão as dependências do nosso projeto. Aqui Podemos ver as versões utilizadas em cada pacote.

Utilizando o Angular Material

Para nosso projeto iremos utilizar o Angular Material (<https://material.angular.io/>). Neste site conseguimos visualizar todos os componentes prontos que ele traz.

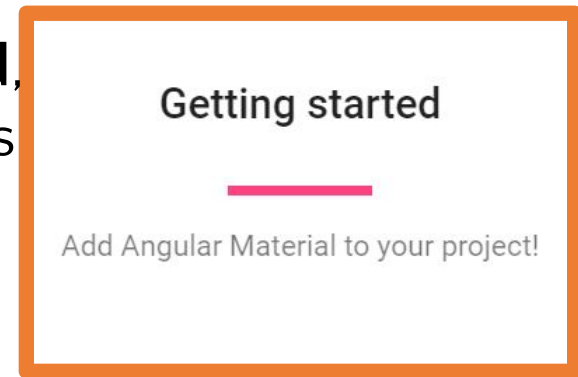


Ao acessar o site, clicando em **Components**, conseguimos visualizar uma lista com todos os componentes que Podemos utilizar.



Escolhendo um componente, ele traz para você um **overview** sobre aquele componente, mostra também a **API**, ou seja, o módulo que teremos de importar para utilizar este componente em nossa aplicação, e por último, alguns **Exemplos** de utilização, para que você tenha uma previa de como ficará este componente na sua aplicação.

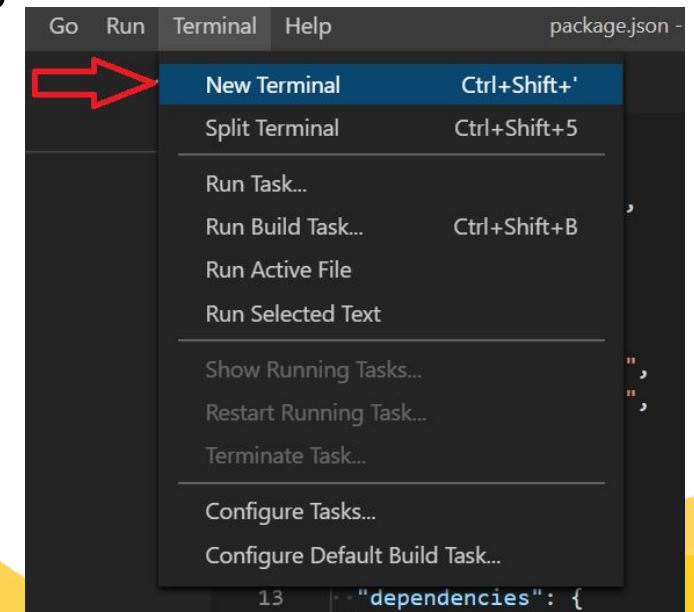
Na página inicial do site do **Angular Material**, clicando em **Getting started**, ele mostrará o que precisamos fazer em nossa aplicação para utilizarmos estes componentes prontos em nossa aplicação.



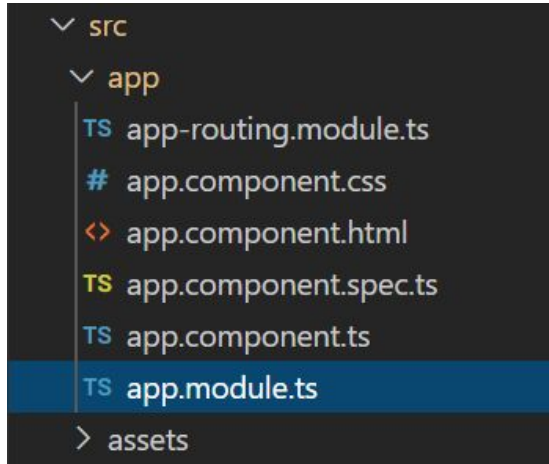
Começaremos instalando o Angular Material utilizando o seguinte comando: **ng add @angular/material**. Após executar este comando, irá aparecer uma mensagem para que você escolha o tema do Angular Material que você deseja utilizar. Para esta aula, iremos escolher a primeira opção: **Indigo/Pink**. Em seguida ele irá perguntar se você deseja utilizar os estilos globais do Angular, vamos utilizar e prosseguir. Em seguida ele irá perguntar se queremos utilizar as animações, aceitamos e prosseguimos.



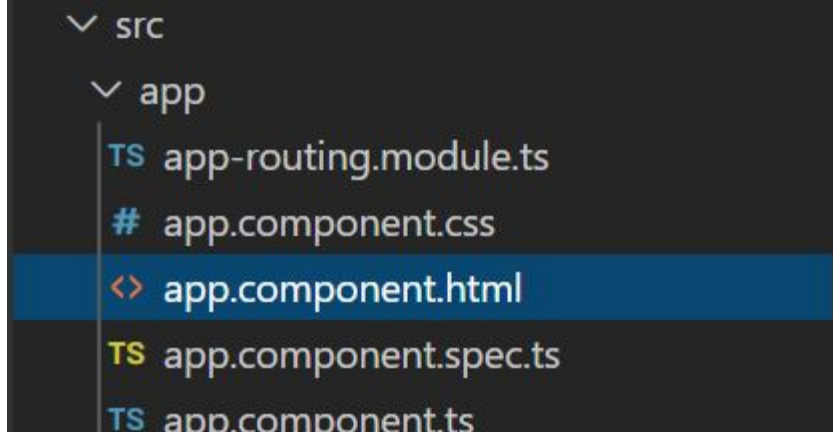
Dica: Clicando em Terminal / New Terminal, você consegue colocar todos os comandos nele e fazer a mesma execução que faria em um terminal externo.



Criando nosso primeiro componente



Dentro de nossa pastinha APP, temos o arquivo: **app.module.ts**, ele é o módulo raiz da nossa aplicação. Sempre que formos utilizar um novo componente, precisamos adicioná-lo neste arquivo para que nossa aplicação passe a enxergá-lo. Tanto os componentes criados por nós, como os que importamos, como por exemplo o do Angular Material.



O arquivo **app.component.html** é aquele arquivo da página inicial do Angular, que foi criado quando iniciamos um novo projeto, contendo a página de Boas vindas do Angular.


```
▼ app  
  TS app-routing.module.ts  
  # app.component.css
```

O arquivo **app-routing.module.ts** é responsável por mapear as rotas, falando para nossa aplicação qual arquivo ele deve abrir.

No arquivo **app.component.css** Podemos definir o estilo que será aplicado para este componente em específico.

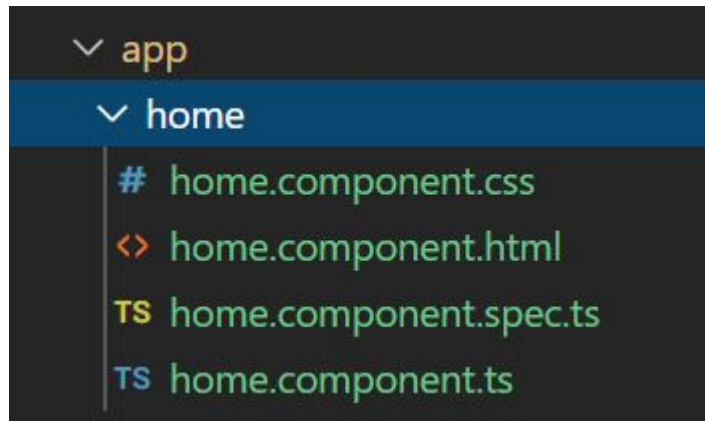


Dica: Iremos criar a aplicação nas pastas raiz do projeto, sem nenhum tipo de estruturação, pois amanhã vocês terão uma aula exclusiva para isso.

Para criarmos nosso primeiro componente, iremos utilizar o comando abaixo:

ng g c home (g = generate / c = component / home = nome do nosso componente).

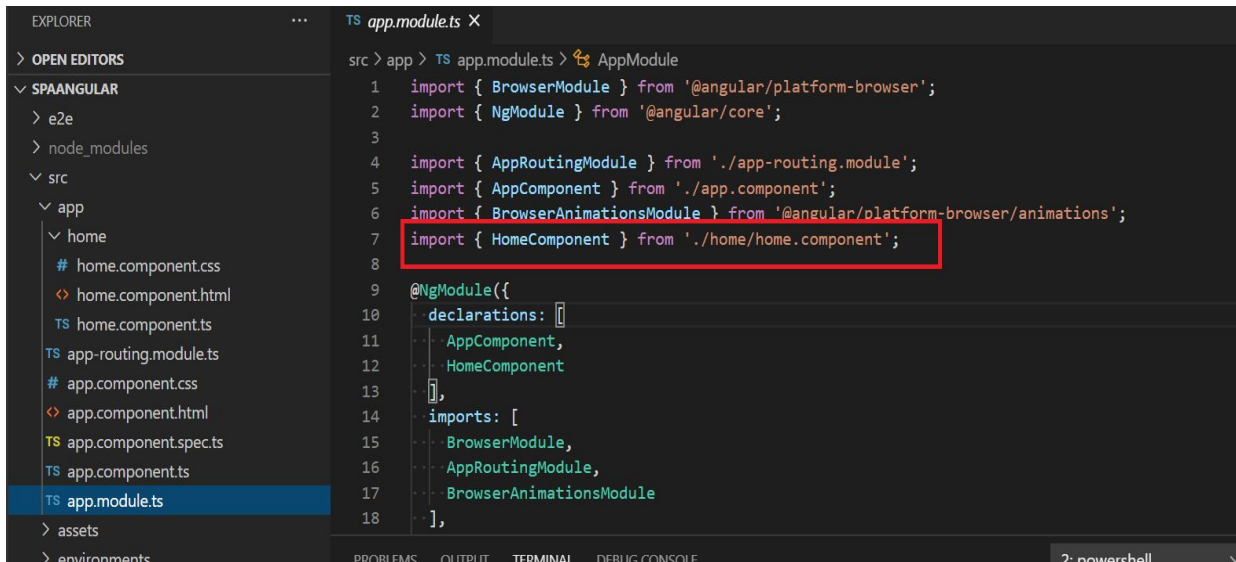
Após este comando, teremos:



Para esta aula não iremos utilizar o arquivo **home.component.spec.ts**, então, podemos excluí-lo, e ficar apenas com os arquivos de CSS, HTML e o Type Script.



Dica: Este arquivo está relacionado aos testes de sua aplicação. Para saber mais acesso o link: (<https://angular.io/guide/testing>)



```
src > app > TS app.module.ts > AppModule
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
7  import { HomeComponent } from './home/home.component';
8
9  @NgModule({
10   declarations: [
11     AppComponent,
12     HomeComponent
13   ],
14   imports: [
15     BrowserModule,
16     AppRoutingModule,
17     BrowserAnimationsModule
18   ],
```

Ao criar nosso módulo, no arquivo **app.module.ts**, é possível ver que ele foi automaticamente importado e inserido nas declarações deste arquivo.



Dica: Utilizando o atalho: CTRL + K + S você consegue visualizar todos os atalhos disponíveis no VSCode.

Definindo nosso módulo como inicial

No arquivo **app-routing.module.ts**, vamos editar sua rota, para que ele abra como default a página **HTML** do componente Home que acabamos de criar. Para isso, vamos inserir o código ao lado:

```
const routes: Routes = [  
  {  
    path: '',  
    component: HomeComponent  
  }  
];
```

Em seguida iremos limpar quase todo o código padrão do nosso arquivo: **app.component.html**, deixando apenas o código: `<router-outlet></router-outlet>`. Este código é responsável por abrir os demais componentes que formos criando dentro do Angular.

Feito isso, execute novamente o comando **ng serve -o** para visualizar nossa nova aplicação.

Adicionando um componente do Angular Material

No site do Angular Material, vamos escolher um componente para adicionarmos em nossa aplicação. Inicialmente irei escolher inicialmente o component Table.

Feito isso, o primeiro passo é importar este novo componente para dentro de nosso projeto. **app.module.ts**, conforme foi feito ao lado:

Feito isso, podemos copiar o HTML e o CSS e colar em nosso componente, para utilizá-lo em nossa aplicação.



Dica: Utilize o atalho: Shift + Alt + F para indentar seu código no Visual Studio Code.

```
app > TS app.module.ts > AppModule
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserModuleAnimationsModule } from '@angular/platform-browser/animations';
import { HomeComponent } from './home/home.component';

import { MatTableModule } from '@angular/material/table';

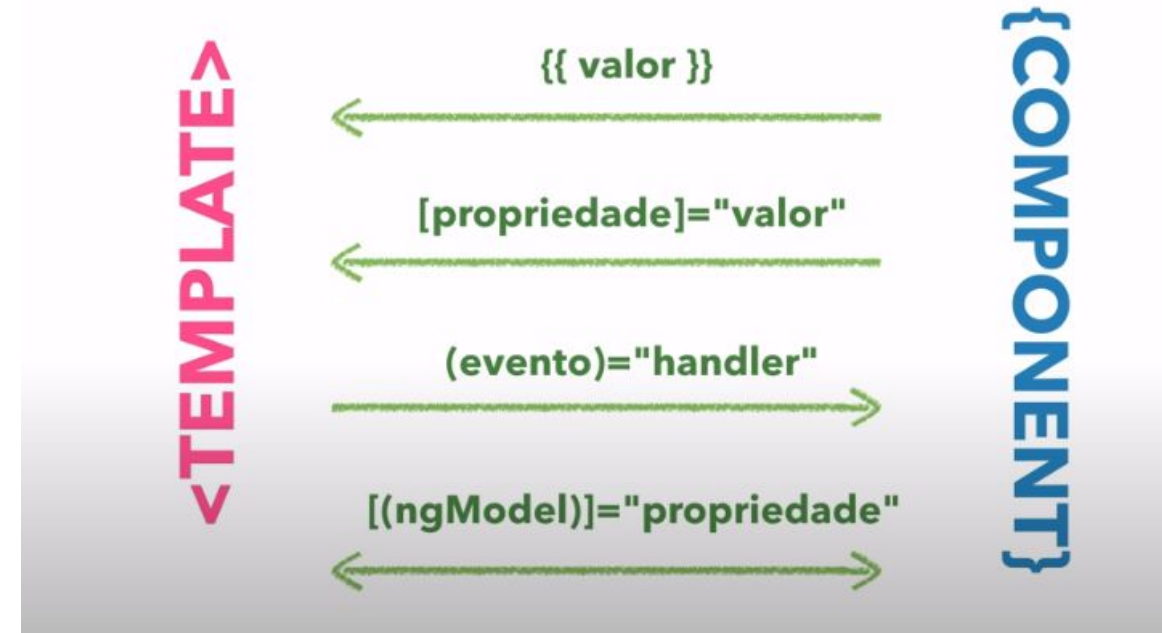
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserModuleAnimationsModule,
    MatTableModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Data Biding

Data Bading é uma forma de associarmos informações que estão no componente para o template e vice versa.

A **Interpolação** nada mais é que um recurso de 'embedar' expressões dentro de uma área delimitada por `{{ .. }}`. Dentro dessa área podemos trazer variáveis e métodos do nosso TS, realizar cálculos matemáticos, e também realizar operações lógicas.

Veja mais a respeito aqui: [Entendendo de vez a interpolação](#) e aqui: [Property Binding e Interpolação](#)



Não existe uma maneira correta de realizar Data Binding, cada situação pode ser usada da forma como quiser e/ou conseguir aplicar. Existem 4 formas:

- **Interpolação:** {{ valor }}

associa informação do componente para o template (HTML)

- **Property Binding:** [propriedade]="valor"

associa informação do componente para o template (HTML)

- **Event Binding:** (evento)="handler"

associa informação do template (HTML) para o componente

- **Two-Way Data Binding:** [(ngModel)]="propriedade"

associa informação entre ambos, ou seja, mantém ambos atualizados (componente e template (HTML)).

Event Biding

O Event Biding consiste em permitir que você escute e responda às ações do usuário, como pressionamentos de tecla, movimentos do mouse e cliques.

Sintaxe: `<button (click)="onSave()">Save</button>`

A associação de evento escuta os eventos de clique do botão e chama o `onSave()` método do componente sempre que ocorre um clique.



Dica: Acesse o link: (<https://angular.io/guide/event-binding>) para entender melhor sobre. Na próxima aula a Geovana falará um pouco sobre também! =]

Diretivas: ngIf / ngFor

As diretivas são marcadores em um elemento DOM (como um atributo) que informam ao Angular para anexar um comportamento especificado a um elemento existente. Existem muitas diretivas prontas que podemos usar e também podemos criar nossas próprias diretivas. Mas hoje iremos ver duas que considero principais

ngIf: Esta é uma diretiva que você adiciona a um elemento na marcação, geralmente para um elemento de contêiner como um div.

ngFor: Esta é uma diretiva para processar cada item de um objeto iterável, gerando uma marcação para cada um. Ela é conhecida como uma diretiva estrutural porque pode alterar o layout do DOM adicionando e removendo elementos DOM de visualização.

Event Emitter

Input Property - @Input()

Você irá utilizá-lo quando quiser receber dados de um componente pai.

Output Property - @Output()

Você irá utilizá-lo quando quiser enviar dados de um componente filho para um componente pai.

Veja mais aqui: [EventEmitter aprendendo a usar Input e OutPut property](#)

Observable

Um Observable, por definição é uma coleção que funciona de forma unidirecional, ou seja, ele emite notificações sempre que ocorre uma mudança em um de seus itens e a partir disso podemos executar uma ação.

Observables e Services (Consumindo uma API)

Entendendo RxJS Observable com Angular

Criando um Service

Comando: `ng g s service/cadastro`

Life Cycle hooks

Cada componente no Angular possui um conjunto de **Eventos de Ciclo de Vida**, eles ocorrem quando o componente é criado, renderizado, tem o valor de suas propriedades alteradas, ou quando ele é destruído. Quando um desses eventos é chamado, o Angular invoca uma série de métodos que são executados imediatamente. Um exemplo de seu uso pode ser aplicado em um site de compras, onde conseguiríamos agir de acordo com cada mudança feita pelo usuário, ao adicionar um item no carrinho.



Atenção: Use com sabedoria para não comprometer a performance de sua aplicação!

OnChanges

Power:
Hero.name:

[Reset Log](#)

Windstorm can sing

-- Change Log --

hero: currentValue = {"name":"Windstorm"}, pre
power: currentValue = "sing", previousValue = {

[back to top](#)

DoCheck

Power:
Hero.name:

[Reset Log](#)

Windstorm can sing

-- Change Log --

OnChanges: hero: currentValue = {"name":"Windstorm"}, previousValue = {}
OnChanges: power: currentValue = "sing", previousValue = {}
DoCheck: Hero name changed to "Windstorm" from ""
DoCheck: Power changed to "sing" from ""
DoCheck called 26x when no change to hero or power

Constructor X NgOnInit: De um modo geral, o constructor vai construir tudo que você precisa antes de iniciar um componente. Já no caso do NgOnInit, ele é chamado toda vez que iniciamos um componente, facilitando assim a chamada de métodos que precisamos que sejam executados assim que o componente for chamado.

Dica: Acesse o link: (<https://angular.io/guide/lifecycle-hooks>) para visualizar uma lista de todos os Life Cycles existentes com exemplos de onde e como eles podem ser aplicados. Saiba um pouco mais a respeito nestes dois vídeos:



Ciclo de vida do Componente

Angular / Components / Lifecycle Hooks



Dicas de conteúdo / Próximos passos

Aprofundando o conhecimento

Dicas de conteúdo

[Tutorial ao Angular
Material](#)

[Tour of Heroes app and
tutorial](#)

[Por que utilizamos
SPA?](#)

[Versão 10, o que tem
de novo?](#)

[Principais dúvidas
sobre o Framework](#)

[Curso de Angular
Grátis no YouTube](#)

[Demos de Layout
Angular](#)

[Blog Oficial do
Angular](#)

[Como atualizar seu
projeto para o Angular
10 + Novidades](#)

[Site para update do
Angular](#)

[Site Oficial do Angular](#)

[Observables e Services
\(Consumindo uma API\)](#)

[Lições que aprendi
como Dev trabalhando
alocada no cliente](#)

[Dicas para se atualizar
em tecnologia através
de Microlearning](#)

Desafio

- 1) Crie uma aplicação Angular (Não esqueça que todo código precisa estar no GitHub)
- 2) Crie no mínimo 3 componentes que deverão estar interagindo na mesma página (Não esqueça das rotas)
- 3) Escolha no mínimo 4 componentes do Angular Material e incorpore eles em sua aplicação.
- 4) Crie em uma das páginas uma interação através da Interpolação e/ou utilizando Property Biding.
- 5) Crie uma explicação sobre o que foi feito, esta explicação será inserida na plataforma junto com o link para o código no **GitHub**.
- 6) Você poderá criar um vídeo demonstrativo da aplicação construída, e publicá-lo no **YouTube**, passando o link de acesso na plataforma. Este passo 6, não é obrigatório, mas entrará como um diferencial.



Dúvidas?

Muito Obrigada!