|  |
| --- |
| **Instalando o Angular:**   1. npm install -g @angular/cli instala a última versão do Angular 2. ng –version mostra a versão instalada do Angular |

Notas para a versão 9 do Angular

O Angular CLI na versão 9 apresenta uma pequena diferença na criação do projeto, pois ele inclui um assistente com duas perguntas básicas, no qual podemos adicionar determinados recursos na hora de criar um novo projeto.

No vídeo seguinte criamos o projeto com uma versão anterior a 9, então o projeto é criado diretamente, mas na versão 9 ele pergunta se queremos adicionar configurações de rotas, e também como queremos os estilos CSS no projeto.

Para nossos projetos aqui criados, vamos utilizar os valores padrão, então na hora da criação do projeto, simplesmente pressione ENTER por duas vezes, uma para cada pergunta, pois assim usaremos os valores padrão do Angular 9 na criação do projeto.

Você verá as seguintes mensagens ao criar o projeto com o Angular CLI 9 nos passos do próximo vídeo:



E:



Ao criar o primeiro projeto, Olá mundo!, você reparará que a tela inicial que o Angular 9 gera é bem mais elaborada e bonita. Você poderá fazer as alterações sugeridas no próximo vídeo do mesmo modo, mas caso deseje ter o mesmo resultado do vídeo, altere o código do arquivo ola-mundo/src/app/app.component.html para:

1. <h1>
2. {‌{title}}
3. </h1>

**Arquivo package.json**

**Comandos que vc pode executar com o próprio angular:**

    "ng": "ng",

    "start": "ng serve",

    "build": "ng build",

    "test": "ng test",

    "lint": "ng lint",

    "e2e": "ng e2e"

**Rodar o projeto a primeira vez: no Terminal na pasta do projeto comando: ng serve --open**

Usamos este comando por que já temos a pasta no esqueleto do projeto que é a node\_modules – que tem todas as bibliotecas que ele baixa com o package json

Se o projeto não tivesse essa pasta primeiro seria necessário o comando: **npm i** pra ele baixar as dependências necessárias

Rodar o projeto normalmente: **ng serve**

Fazer build: **ng build**

**Entendendo a estrutura de pastas do projeto:**

**package.json:** gerenciar as dependências do projeto (similar ao pom.xml do maven)

**package-lock.json:** grava o último npm i que você fez (Não é obrigatório) não é legal ser versionado, então colocar no gitignore ou apagar

**tsconfig.json:** serve pra dizer que o TypeScript que vamos escrever não é interpretado pelo navegador e sim convertido para JavaScript e esse arquivo convertido sim, é o que vai para o navegador, este arquivo que vai para o navegador é criado ao fazer o build do projeto dentro da pasta dist que é criado na hora do build também.

**tslint.json:** Pra ele achar alguns erro importantes no seu código, alguma coisa duplicada, alguma coisa do angular ou variável que possa dar erro

**pasta src:** Onde vc vai trabalhar de fato, tem os seguintes itens:

index.html: página inicial padrão, a princípio não mexemos nela

pasta app:

app.module.ts: serve para carregar todos os módulos do Angular, devido ao formato do Angular de single page aplication – que é gerenciar o sistema inteiro com apenas uma página, carregando os componentes dele, que são classes que você escreve em TypeScript e que você pode criar vários componentes para renderizar no html

app.component.html: html do componente inicial

app.component.spec.ts: teste do componente pra saber se ele existe

app.component.css: css do componente

app.component.ts: componente em si

**Criar módulo:** ng generate module **nomeDoModulo ou** ng g module **nomeDoModule**

**Criar componente:** ng generate component **nomeDoComponente ou** ng g component **nomeDoModuloQueEleFaraParte/nomeDoComponente**

Ao rodar esse comando ele cria dentro da pasta app, uma pasta com o nome do componente e os 4 arquivos padrão.

O componente é uma subdivisão do módulo, um item dele. Intermediação entre a view e a regra de negócio, similar ao controller. Ele gera uma tag html.

**Criar service:** ng generate service **nomeDoModuloQueFicaraDentro/services(cria este diretório caso ele já não exista/nomeDoService ou** ng g service **nomeDoModuloQueFicaraDentro/services(cria este diretório caso ele já não exista/nomeDoService**

Para guardar a lógica de negócio

Property Binding – propriedade do componente que passa para o html

Event Binding – Você passa uma chamada do html para o seu componente para ele ter algum comportamento

**Imagens:** Para usar imagens é necessário inserir na pasta assets

**Criar módulo de rotas:** **ng generate module app-routing --flat --module=app**

No comando a cima nós estamos gerando um novo módulo chamado AppModuleRouting e passando os parâmetros: -flat para que ele seja criado dentro da pasta src/app e o -module=app para registrarmos ele no nosso arquivo AppModule.

Após criar o módulo de rotas deixar arquivo conforme abaixo:

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

import { CommonModule } from '@angular/common';

import { Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { UsuarioComponent } from './usuario/usuario.

\*\*IMPORT DO COMPONENTE

const routes: Routes = [

  { path: **'nomePathComponent'**, component: **NomeDoComponente** },

{ path: 'home', component: HomeComponent },

  { path: 'usuario', component: UsuarioComponent },

  { path: '', pathMatch:'full', redirectTo:'/home' },

];

@NgModule({

  declarations: [],

  imports: [

    RouterModule.forRoot(routes) \*\*Import das suas rotas

  ],

exports: [RouterModule]

})

export class AppRoutingModule { }

detalhes: <https://angular.io/tutorial/toh-pt5>

Associar módulo, ao módulo principal da aplicação:

No arquivo app.module.ts fazer conforme abaixo:

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

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

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

import { CalculadoraModule } from './calculadora/calculadora.module';

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    CalculadoraModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

também é possível tornar o módulo calculadora público dentro da pasta calculadora, desta forma, só é necessário fazer o import da “pasta”, não precisa do nome do módulo, pra isso, basta na pasta calculadora, criar o arquivo index.ts conforme abaixo:

export \* from './calculadora.module';

depois, se voltar no app.module.ts e retirar o module do nome, tudo continuar compilando como antes, veja abaixo:

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

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

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

import { CalculadoraModule } from './calculadora';

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    CalculadoraModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

Este recurso é opcional

Notas para a configuração do Bootstrap no Angular 9

Conforme mencionado anteriormente, caso você esteja usando o Angular 9, a importação do arquivo CSS do Bootstrap deverá ser adicionado com a seguinte estrutura ao arquivo "angular.json":

1. ...
2. "styles": [
3. "src/styles.css",
4. "node\_modules/bootstrap/dist/css/bootstrap.min.css"
5. ],
6. ...

Siga os passo da aula seguinte, mas adicione o CSS conforme mencionado acima para o Angular 9.

**Comando** npm install --save bootstrap@3

O @3 é pra garantir que a versão que será baixada é a 3.

**Implementando a interface do CalculadoraComponent**

Como fazer o botão de resultado capturar o resultado do get display():

Neste caso o atributo value já é do próprio input, usando os conchetes dizemos ao Angular que queremos fazer um parse desse elemento, o operador igual diz qual método get do componente com esse nome será chamado para esse bind, que nesse caso é o display, desta forma o value estará sempre em sincronismo com o display

<input

                  type="text"

                  class="form-control text-right kz-input-text"

                  id="display"

                  name="display"

                  [value]="display"

                  readonly>

Como atribuir valor aos números e operadores da calculadora:

Como é um botão primeiro dizemos para o angular que será acionada uma ação/evento quando houver um click usando o (click) depois do operador igual dizemos entre aspas duplas qual o evento/ação será chamado ao clicar no botão, neste caso estamos chamando o método adicionarNumero passando como parâmetro para o método o número em questão, usando aspas simples para não dar conflito com as aspas duplas do método em si

                <button

                  type="button"

                  class="btn btn-default kz-btn"

                  (click)="adicionarNumero('7')"

                  id="btn7">7

                </button>

**Projeto 3:**

**Configurando as rotas de navegação no componente principal**

Primeira coisa, criar um arquivo dentro da pasta app que será um module responsável por gerenciar as rotas.

Arquivo: app-routing.module.ts

Conteúdo:

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

import { Routes, RouterModule } from '@angular/router';

export const routes: Routes = [

];

@NgModule({

    imports: [ RouterModule.forRoot(routes) ],

    exports: [ RouterModule ]

})

export class AppRoutingModule {}

Depois importar o arquivo de rotas para o module principal.

Para indicar onde as rotas vão aparecer na tela, colocar a tag abaixo no html em questão. Nesse caso vai ficar embaixo da barra de navegação, que colocamos no app.component.html pois ela será fixa para todas as páginas:

<router-outlet></router-outlet>

Criar o módulo de tarefas

Criar arquivo index.ts para facilitar os imports que venham do módulo de tarefas

Conteúdo: export \* from './tarefas.module';

Importar o módulo criado no app.module.ts

Por convenção caso tenha um arquivo que vai ser usado em muitos lugares criamos uma pasta shared dentro do módulo em questão (neste caso tarefas) e deixamos ou criamos os arquivos compartilhados lá dentro.

Criar o model de tarefas

Criar novo arquivo index.ts dentro de shared para os imports

Atualizar o arquivo index.ts do tarefas incluindo o export do shared

Criar service de tarefas (dentro do diretório shared)

Ng g service tarefas/shared/tarefa

Atualizar arquivo index.ts do model com o tarefa.service

Fazer o import do serviço no tarefas.module.ts

Importante: Como é um serviço no NgModule ele é inserido como um providers

Exemplo: providers: [ TarefaService]

No tarefa.service fazer o import de tarefa e depois a implementação da lógica

Criar componente listar-tarefa: ng g componente tarefas/listar-tarefa

Criar novo arquivo index.ts dentro de listar para os imports

Atualizar o arquivo index.ts do tarefas incluindo o export do listar

Criar o arquivo tarefas-routing.module.ts para mapear as rotas do componente

Conteúdo:

import { Routes } from '@angular/router';

import { ListarTarefaComponent } from './listar';

export const TarefaRoutes: Routes = [

    {

        path: 'tarefas',

        redirectTo: 'tarefas/listar'

    },

    {

        path: 'tarefas/listar',

        component: ListarTarefaComponent

    },

];

Nota para a criação da interface HTML

Alguns navegadores poderão não renderizar a tabela como demonstrado no vídeo a seguir ao listar as tarefas, caso isso ocorra, adicione o elemento <tbody> a ela conforme o código a seguir:

1. <table class="table table-striped table-bordered table-hover">
2. <tbody>
3. ...
4. </tbody>
5. </table>

Tente executar os passos do vídeo a seguir normalmente, mas tenha em mente que se ouvir um problema de visualização, retorne a essa nota e faça a alteração demonstrada acima para corrigir o problema.

Criar a implementação do ComponenteListar

Diretiva – É um bloco de código específico que adiciona um comportamento ou alguma regra de negócio para um componente. Gera um atributo para um componente.

Criar o componente CadastrarTarefa e repetir os passos da criação do componente Listar

Nota sobre o ViewChild do vídeo a seguir

No vídeo a seguir introduziremos um novo elemento, o ViewChild, que será explicado em detalhes na aula.

No Angular 9 devido a algumas alterações no processo de compilação do mesmo, foi adicionado um segundo parâmetro ao elemento @ViewChild que utilizaremos a seguir.

Portanto se você estiver acompanhando o curso com o Angular 9, no vídeo a seguir quando implementar a seguinte linha de código:

@ViewChild('formTarefa') formTarefa: NgForm;

Tenha certeza de incluir o novo atributo, ficando da seguinte forma:

@ViewChild('formTarefa', { static: true }) formTarefa: NgForm;

Crio uma variável local do form para ser usada em outros lugares do HTML:

<form #formTarefa="ngForm">

Criar o componente EditarTarefa e repetir os passos da criação do componente Listar

Removendo tarefa e atualizando tarefa tudo no componente Listar

Criar uma diretiva: ng g directive tarefas/shared/tarefa-concluida

A Diretiva é como se fosse um componente, mas ao invés de gerar uma tag hml ela gera um atributo html. Usada quando você quer gerar uma ação específica em um componente. Serve para aplicar um comportamento, também serve para criar máscaras para campos de texto ou alterações específicas em um determinado componente.

Nota importante!!!

Nesse projeto utilizamos uma API para a conversão de moedas, a **https://fixer.io**, que mudou a forma como acessamos os dados, portanto será necessário modificar a URL da API no código do serviço que será criado a seguir para que ela continue funcionando.

A nova API requer uma chave de acesso pessoal. Para facilitar a implementação eu criei uma pessoal e compartilharei com você, assim não haverá a necessidade de você criar a sua.

**\* Essa chave tem uma limitação que somente permite converter moeda de Euros (EUR) para outras moedas, então ao executar a aplicação, tenha certeza de selecionar no campo "De" sempre a moeda EUR, pois as demais não funcionarão por limitação da conta gratuita que eles disponibilizam.**

**\*\* Caso você deseje utilizar essa API em outros projetos, peço que crie a sua própria chave de autenticação no website citado acima, pois essa também possui uma limitação de 1000 requisições mensais, que deverá ser compartilha somente aqui pelos alunos.**

A versão 9 do Angular também traz uma nova API para requisições HTTP, que é a HttpClient (introduzida no Angular 4), que veio para substituir a antiga Http.

A seguir segue o código completo do serviço atualizado para a versão 9 do Angular, assim como para a nova API do fixer.io.

Você pode acompanhar normalmente as duas próximas aulas sobre a criação do serviço, substituindo o código antigo pelo novo que segue abaixo:

1. import { Injectable } from '@angular/core';
2. // Importamos o HttpClient ao invés do Http para o Angular 7
3. //import { Http } from '@angular/http';
4. import { HttpClient } from '@angular/common/http';
5. import { Observable } from 'rxjs/Observable';
6. import {
7. Conversao,
8. ConversaoResponse
9. } from '../models';
10. @Injectable()
11. export class ConversorService {
12. // Nova url do fixer.io, que adiciona o parâmetro access\_key, que é a chave de autenticação
13. //private readonly BASE\_URL = "http://api.fixer.io/latest";
14. private readonly BASE\_URL = "http://data.fixer.io/api/latest?access\_key=eba7130a5b2d720ce43eb5fcddd47cc3";
15. constructor(private http: HttpClient) {}
16. /\*\*
17. \* Realiza a chamada para a API de conversão de moedas.
18. \*
19. \* @param Conversao conversao
20. \* @return Observable<ConversaoResponse>
21. \*/
22. converter(conversao: Conversao): Observable<any> {
23. // Na linha abaixo altere a '?' por '&'
24. let params = `&base=${conversao.moedaDe}&symbols=${conversao.moedaPara}`;
25. return this.http
26. .get(this.BASE\_URL + params);
27. // No Angular 6 as duas próximas linha não são mais necessárias
28. //.map(response => response.json() as ConversaoResponse)
29. //.catch(error => Observable.throw(error));
30. }
31. /\*\*
32. \* Retorna a cotação para dado uma response.
33. \*
34. \* @param ConversaoResponse conversaoResponse
35. \* @param Conversao conversao
36. \* @return number
37. \*/
38. cotacaoPara(conversaoResponse: ConversaoResponse,
39. conversao: Conversao): number {
40. if (conversaoResponse === undefined) {
41. return 0;
42. }
43. return conversaoResponse.rates[conversao.moedaPara];
44. }
45. /\*\*
46. \* Retorna a cotação de dado uma response.
47. \*
48. \* @param ConversaoResponse conversaoResponse
49. \* @param Conversao conversao
50. \* @return string
51. \*/
52. cotacaoDe(conversaoResponse: ConversaoResponse,
53. conversao: Conversao): string {
54. if (conversaoResponse === undefined) {
55. return '0';
56. }
57. return (1 / conversaoResponse.rates[conversao.moedaPara])
58. .toFixed(4);
59. }
60. /\*\*
61. \* Retorna a data da cotação dado uma response.
62. \*
63. \* @param ConversaoResponse conversaoResponse
64. \* @return string
65. \*/
66. dataCotacao(conversaoResponse: ConversaoResponse): string {
67. if (conversaoResponse === undefined) {
68. return '';
69. }
70. return conversaoResponse.date;
71. }
72. }

O HttpClient citado acima pertence a outro módulo no Angular 9, que deverá ser modificado também no arquivo 'app.module.ts', conforme código a seguir:

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule } from '@angular/forms';
4. // Importe o módulo HttpClientModule ao invés de HttpModule
5. //import { HttpModule } from '@angular/http';
6. import { HttpClientModule } from '@angular/common/http';
7. import { AppComponent } from './app.component';
8. import { ConversorModule } from './conversor';
9. @NgModule({
10. declarations: [
11. AppComponent
12. ],
13. imports: [
14. BrowserModule,
15. FormsModule,
16. // Modifique aqui a importação do módulo também
17. //HttpModule,
18. HttpClientModule,
19. ConversorModule
20. ],
21. providers: [],
22. bootstrap: [AppComponent]
23. })
24. export class AppModule { }

Com isso você terá o serviço atualizado e funcionando.

Nota sobre o ViewChild do vídeo a seguir

No Angular 9 devido a algumas alterações no processo de compilação do mesmo, foi adicionado um segundo parâmetro ao elemento @ViewChild que utilizaremos a seguir.

Portanto se você estiver acompanhando o curso com o Angular 9, no vídeo a seguir quando implementar a seguinte linha de código:

@ViewChild("conversaoForm") conversaoForm: NgForm;

Tenha certeza de incluir o novo atributo, ficando da seguinte forma:

@ViewChild("conversaoForm", { static: true }) conversaoForm: NgForm;

**\*\* Importante:**

no código do vídeo a seguir criaremos 4 atributos de classe como sendo **private**. Na versão 9 do Angular a palavra chave **private** deverá ser *removida* da classe para que os atributos funcionem corretamente, portanto onde você ver o código:

1. private moedas: Moeda[];
2. private conversao: Conversao;
3. private possuiErro: boolean;
4. private conversaoResponse: ConversaoResponse;

Remova o acesso **private**, ficando da seguinte forma:

1. moedas: Moeda[];
2. conversao: Conversao;
3. possuiErro: boolean;
4. conversaoResponse: ConversaoResponse;