Desenvolvimento Web com TypeScript e Angular

Celso Marigo Junior

Engenheiro de Computação, programador **fullstack**, +15 anos de experiência atuando em programação de sistemas **Web** e **Desktop**. Proprietario da empresa /DEV - desenvolvimento Multiplataforma.

cm.junior@gmail.com

Programação

Aula 1 - Introdução (Web, API...);
Aula 2 - Pipes, Diretivas, Services, DI;
Aula 3 - Rotas, Guards, RxJS;
Aula 4 - Ciclos de Vida e Estados da aplicação;
Aula 5 - Environment, deploy, segurança;

Projeto: Sistema Web like Deliveroo

Recursos

https://github.com/cmjunior/curso-angular

https://drive.google.com/drive/folders/1MqaQOhe2KBmcwdN-ws7Rq3FOpvfo5_wx?usp=sharingcurso

Aplicações Web

Frontend
HTML, **CSS** e **JS**
(bootstrap e jquery)



Backend
PHP (Laravel), **NodeJS**
MySQL, MongoDB

TypeScript *by Microsoft*

Supertipo do Javascript(js);

JS orientado a Objetos;

Tipagem Estática x Dinâmica;

Namespaces, decorators.

O que é  ?

Componentes UI

Framework JS mantido pelo Google.
Open Source!

Gerador de Código

Gerador e empacotador de código

Aplicações PWA

Progressive Web Apps (aplicativo web progressivo) [Site > Aplicativo]

SSR

Server-side Render. Site gerado no servidor. Conteúdo estático.

Ferramentas

Angular Cli: **ng**

Formulários Reativos

Rotas

Cliente HTTP

Animações

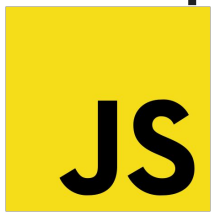
mais...

"**npm** (node package manager) possui aproximadamente **10 milhões** de usuários que baixam mais de **30 bilhões** de pacotes por mês"

Cross-Platform

Desenvolvimento de aplicações multiplataforma, simples ou complexas!

JavaScript



HTML

CSS

Webpack compacta versões diversas de builds para cada browser.

Pré Requisitos

Ferramentas para desenvolver em Angular

https://nodejs.org/

Visual Studio Code

https://code.visualstudio.com

Angular CLI (ng)

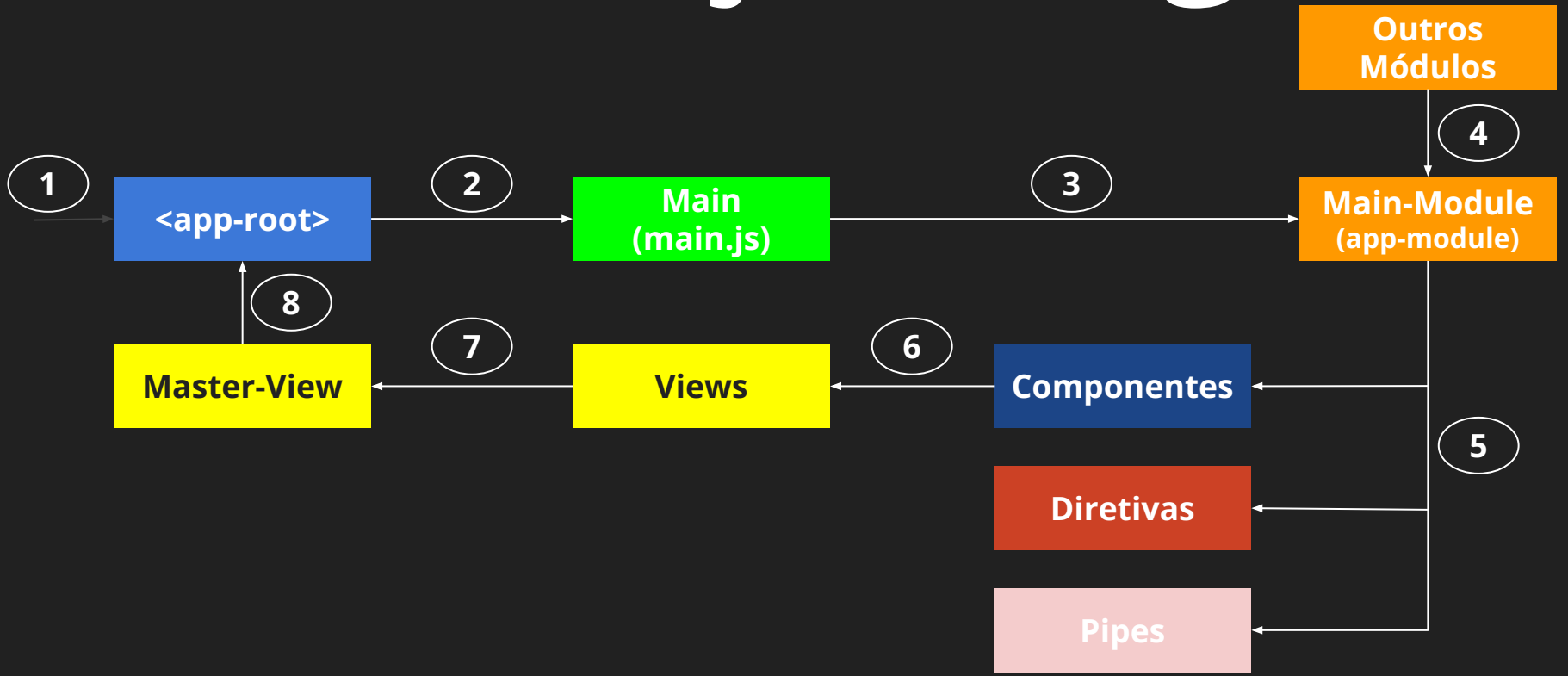
Poderosa ferramenta de linha de comando para geração de código angular

```
npm install -g @angular/cli

ng new meu-projeto

cd meu-projeto

ng serve
```

Blocos Angular

- Módulos
- Componentes
- Templates
- Metadata
- Data Binding
- Diretivas
- Pipes
- Rotas
- Serviços
- Injeção de Dependência

Inicialização Angular



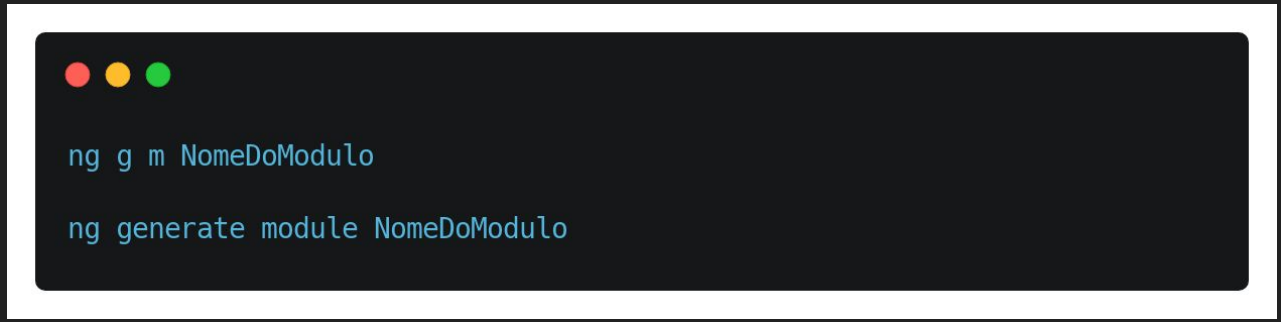
Módulos

Módulos organizam a aplicação em blocos por funcionalidade;

Um módulo angular, é uma classe decorada com **@NgModule**;

Recebe como parâmetro um objeto cujas propriedades descrevem o módulo;

Cada aplicação angular deve ter ao menos um módulo, este é conhecido como *root-module*



ngModule (metadata)

imports: Contém as dependências do módulo, dependência de todos os componentes declarados neste módulo;

declarations: Componentes, Diretivas e Pipes que pertencem ao módulo;

bootstrap: A view principal do módulo, apenas o root-module pode ter essa propriedade;

exports: Componentes, Diretivas e Pipes, ou módulos importados, que podem ser usados por outros módulos. O root-module, não tem essa propriedade;

providers: Define os serviços fornecidos por este módulo, acessível a seus componentes.

App x Shared Module

```
import ...

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    MainComponent,
    TrocaSenhaComponent,
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule
    AppRoutingModule,
    SharedModule.forRoot(),
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Decorators / Metadata

- Metadata: dizem ao angular como processar uma classe;

- Decorators: são usados para adicionar metadata a uma classe, inicam com @

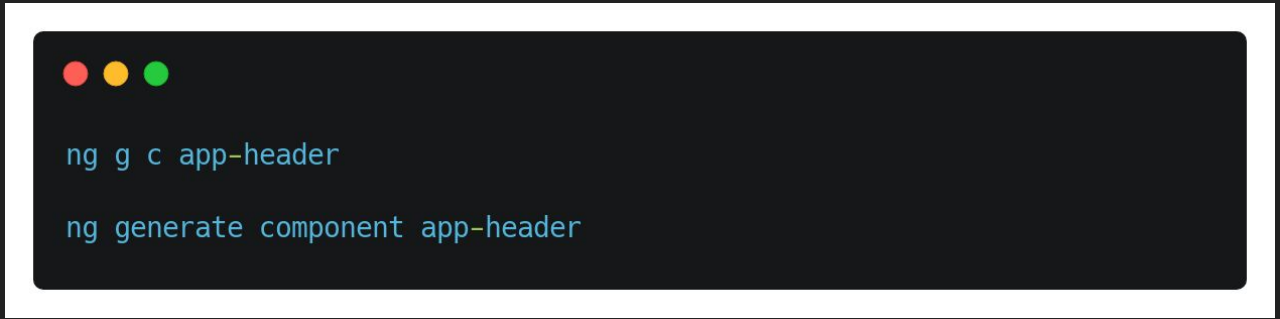
```
import ...

@NgModule({
  declarations: [
    ProfileComponent,
    ListFilterPipe,
  ],
  imports: [
    CommonModule,
    MaterialModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule,
  ],
  exports: [
    ProfileComponent,
    MaterialModule,
    FormsModule,
    ListFilterPipe,
  ],
})
export class SharedModule {
```

Componentes

Componentes são objetos com template (HTML), estilo (CSS) e lógica (TS);

Exportados como elementos HTML customizados, que são identificados por tags, por exemplo: **<app-header></app-header>**;

São inicializados pela engine de injeção de dependências do Angular, sendo acessíveis pelos componentes do mesmo módulo, ou dependentes.



Exemplo

```ts
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'field-box',
  templateUrl: './field-box.component.html',
  styleUrls: ['./field-box.component.scss']
})
export class FieldBoxComponent {
  @Input() label: string
  @Input() value: string

  @Output() clicked = new EventEmitter<any>();

  onClicked() {
    this.clicked.emit()
  }
}
```

```html
<div fxFill (click)="onClicked()" fxLayout="column" class="field-box">
  <div fxLayout="row" fxLayoutAlign="start" class="field-label-container">{{ label }}</div>
  <div fxLayoutAlign="center center" class="field-value-container">{{ value }}</div>
```

```scss
x 0 rgba(0, 0, 0, 0.2);
font-size: 11pt;
border-bottom-left-radius: 6px;
border-bottom-right-radius: 6px;
}
}
```

```html
<field-box label="E-MAIL" value="jose.silva@teste.com" (clicked)="onItemClicked()">
```

@Input / @Output

- @Input: permite que o componente receba valres;

- @Output: permite que o componente emita uma ação, respondendo a um evento, aqui click, passa a ação para o componente pai;

E-MAIL

jose.silva@teste.com

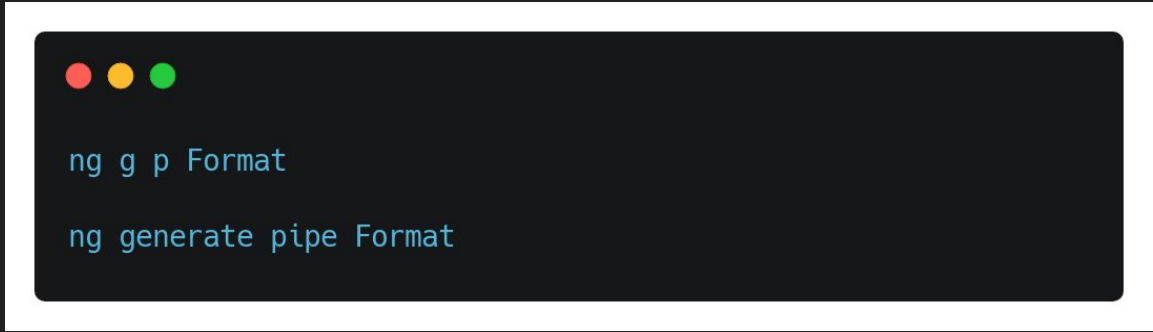
Pipes

Utilizados para modificar, formatar valores em tempo de exibição;

Aceitam argumentos, e pode ser encadeados;

Ao Angular possui alguns pipes nativos, ex: uppercase, date e async;

Podemos criar pipes customizados.



Exemplo

```ts
import { Pipe, PipeTransform } from '@angular/core';

var VMasker = require('vanilla-masker')

export const CPF_FORMAT = '999.999.999-99'
export const CNPJ_FORMAT = '99.999.999/9999-99'

@Pipe({
  name: 'format'
})
export class FormatPipe implements PipeTransform {

  transform(value: string, format?: string): unknown {
    if ( value && value.trim() != '' ) {
      return VMasker.toPattern(value,
        (format ? format : (value.trim().length > 11 ? CNPJ_FORMAT : CPF_FORMAT)));
    }

    return value
  }

}
```

Pipe - Format

- Pipe utilizado para colocar máscara em valores;

- O parâmetro format, da função transform é opcional, caso não passado considera que se deseja formatar um CPF ou CNPJ;

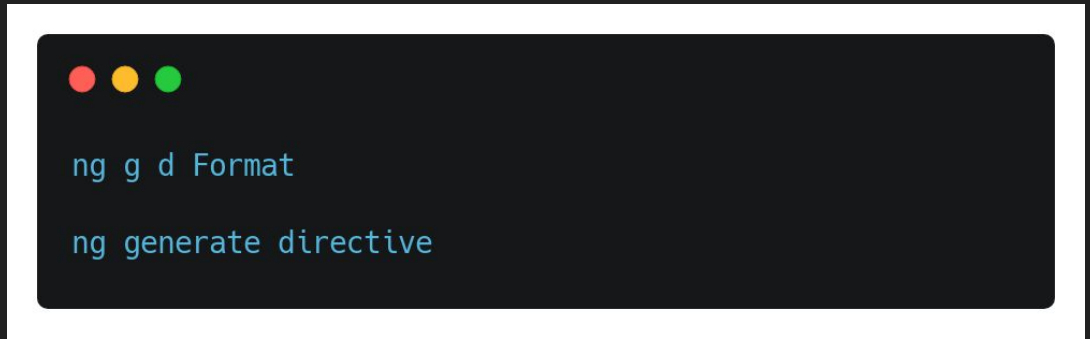
```ts
//ts
let documento = '12345678900'

// html
<span>{{ documento | format }}</span>

// resultado
// 123.456.789-00
```

Diretivas *(directives)*

Componentes são diretivas, com um template;

Diretivas estruturais, modificam, adicionando ou removendo, elementos do DOM, por exemplo ***ngIf**, ***ngFor** e ***ngSwitch** o (*) indica este tipo de diretiva;

Diretivas de Atributos, modificam a aparência ou comportamento de um elemento ou de outro componente angular, por exemplo **[ngStyle]** e **[ngClass]**.



Exemplo

```
import { Directive, Input, HostListener, Renderer2 } from '@angular/core';

@Directive({
    selector: '[enterTab]'
})
export class EnterTabDirective {
    @Input() enterTab

    constructor(private renderer: Renderer2) { }

    @HostListener('keydown', ['$event']) onKeyDown(e) {
        if ( e.which == 13 || e.keyCode == 13 ) {
            let nextEl = this.renderer.selectRootElement(this.enterTab, true)
            e.preventDefault();
            if ( nextEl ) {
                nextEl.focus()
            }
        }

        return
    }
}
```

Directive - EnterTab

- Modifica o comportamentos dos inputs para, ao teclar <ENTER> ir para próximo componente;

- A diretiva, recebe um @Input, com o componente para o qual deve ir, convenientemente, o nome do seletor da diretiva, é o mesmo do parâmetro;

```
<h4 mat-subheader>Acesso ao Sistema</h4>
<form [formGroup]="loginForm">
  <mat-form-field class="login-card-fields">
      <input [enterTab]="password" matInput
formControlName="username"
        placeholder="Informe o Usuario">
  </mat-form-field>
  <mat-form-field class="login-card-fields">
      <input [enterTab]="acessar" #password matInput
formControlName="password"
        placeholder="Informe a Senha">
  </mat-form-field>
</form>
<button #acessar mat-stroked-button
(click)="login()">Acessar</button>
```

Serviços *(services)*

Contém a lógica de negócio da aplicação, e pode gerenciar dados compartilhados pelos componentes;

No geral são os serviços que fazem a comunicação com o backend;

As classes services pode ser injetadas nos componentes, bastando ser declaradas em seus construtores.

Services Singleton

As *services* são decoradas com o decorator **@Injectable**. Isso permite que o core do angular seja capaz de **injetar** a mesma dentro de qualquer componente através de seu construtor.

Utilizando o Angular CLI para gerar uma service, a mesma vem anotada com **@Injectable({ providedIn: 'root' })**, isso faz com que a service seja provida pelo ***root-module***, tornando a mesma um ***singleton***;

Em alguns casos, a service será usada apenas em um módulo específico, nesse caso, precisamos declarar a mesma dentro do módulo dentro da propriedade ***providers***.

Exemplo

UIService

- Serviço utilizado para exibir um spinner;

- Qualquer componente que queira mostrar um spinner, somente precisa injetar a classe no construtor, uma vez que o código abaixo está no app-root;

```
import { MatDialog } from '@angular/material/dialog';
import { Injectable, Inject } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class UiService {
  private _showingProgressBar = new BehaviorSubject<boolean>(false)

  get showingProgressBar() {
    return this._showingProgressBar.asObservable()
  }

  showProgressBar(show: boolean) {
    this._showingProgressBar.next(show)
  }
}
```

```
import { NgxSpinnerService } from 'ngx-spinner';
import { UiService } from './services/ui-service';
...

  constructor (
    private spinner: NgxSpinnerService,
    private uiService: UiService
  ) {}

  ngOnInit(){
    this.uiService.showingProgressBar.subscribe( show => {
      if ( show ) {
        this.spinner.show();
      } else {
        this.spinner.hide();
      }
    })
  }
```

Rotas

Responsável pelo roteamento das páginas;

O Angular utiliza o conceito **SPA (Single Page Application)**, todas as rotas são direcionadas para index.html;

A navegação é feita pelo componente de rotas, que utiliza a **API do browser** para integrar com os botões do navegador.

Desta forma é uma boa prática desenvolver sistemas complexos com navegação baseada em rotas.

Exemplo

```
import { LoginGuard } from './login/login.guard';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LoginComponent } from './login/login.component';

const routes: Routes = [
  { path: '', redirectTo: 'mapas', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'login/:redirect', component: LoginComponent },
  { path: 'mapas',
    loadChildren: () => import('./filter-map/filter-map.module').then( m => m.FilterMapModule ),
canActivate: [LoginGuard] },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

[forRoot() / forChild()]

forRoot(): é o método que usamos para registrar as rotas no root-module;

forChild(): utilizamos nos módulos que são carregados utilizando a técnica de **lazy-loading**, carregamento tardio;

Utilizamos o lazy-loading para que os componentes sejam carregados apenas quando suas rotas forem acionadas;

Outro detalhe a observar é a rota para 'mapas', que tem uma propriedade: **canActivate**. Esta utiliza uma service para verificar se a mesmas pode ser ativada;

Exemplo

```
import { Injectable } from "@angular/core";
import { AngularFireAuth } from '@angular/fire/auth';
import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot }
    from '@angular/router';
import { map } from 'rxjs/operators';

@Injectable({
    providedIn: 'root'
})
export class LoginGuard implements CanActivate {
    constructor(
        private afAuth: AngularFireAuth,
        private router: Router
    ) { }

    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot)
    {
        return this.isAuthenticated(state.url)
    }

    isAuthenticated(path) {
        return this.afAuth.authState.pipe(
            map( user => {
                if ( !user ) {
                    return this.router.parseUrl(`/login/${btoa(path)}`)
                }

                return true
            })
        )
    }
```

LoginGuard

- A service implementa a classe **canActivate**, de **@angular/router**, a mesma verifica se o usuário está logado, caso não esteja redireciona o mesmo para login;

Observables

Objetos que podem retornar múltiplos valores com o tempo, podendo esse comportamento ser cancelado com um unsubscribe;

Podem ser sincronizados (synchronous) ou não-sincronizados (asynchronous);

Nada acontece sem um observer;

Separa uma cadeia de processamento e transformação da inscrição (subscription);

Pode ser reusado ou retornado.

Observable é uma função que recebe um observer;

Um Observable só produz valores no subscribe()

```
saveFluxos(body) {
    return this.httpClient.post(`${this.BASE_URL}/postFluxos`, body.toString(), HTTP_OPTIONS)
}

...
// Incorreto
this.saveFluxos(body)

...
// Correto
this.saveFluxos(body).subscribe( result => {
    console.log('Tudo Salvo')
}, (error) => {
  console.log('Ocorreu um erro')
})
```

async pipe

Podemos passar um *Observable* diretamente para o *async* pipe;

Automaticamente faz o unsubscribe;

Utilizando o "*async as*" atribuímos o resultado do observable a uma variável.

BehaviorSubject protegido com private na service, pois apesar de ter um estado, a forma segura de obter seu valor é através do observable.

```
// service
private _clientes = new BehaviorSubject<Cliente[]>([])

get clientes() {
  return this._clientes.asObservable()
}

loadClientes() {
  this.httpClient.get<Cliente[]>(`${this.URL}/clientes`).subscribe( clientes =
    this._clientes.next(clientes)
  }, (error) => {
    this.uiService.showError('Erro ao obter clientes', error.message)
  })
}

// Componente
@Component({
  selector: 'clientes',
  template: `
  <ng-container *ngIf="clientes$ | async as clientes; else carregando">
    <app-clientes [clientes]="clientes"></app-clientes>
  </ng-container>
  <ng-template #carregando>
    <mat-spinner></mat-spinner>
  </ng-template>
  `
})
export class ClientesComponent(
  clientes$ = this.clientesService.clientes;
  ...
}
```