front-end

prof. Lucas Ferreira

engenharia de software

engenharia de computação

Frameworks de Interação + Angular

SPA

Single Page Applications

- Aplicações completas, desenvolvidas (normalmente) em JavaScript, que funcionam quase como se estivessem sendo executadas nativamente no desktop
- Aplicações modernas que necessitam de uma performance elevada com UIs fluídas
- Quando mudamos de links ou seções em um app SPA, a mudança é quase instantânea, não existe load time complexo
- Porém podemos ter um load time maior quando abrimos pela primeira vez (maior payload)
- Objetiva evitar o recarregamento da aplicação em cada mudança de tela e também diminuir a quantidade de requisições ao servidor
- O **Gmail** é um exemplo óbvio de uma SPA que usamos frequentemente

SPAs x Frameworks de Interação

Dado a necessidade criação de interfaces do usuário (UI) cada vez mais complexas e escaláveis que devem responder de forma fluída a interação dos usuários, o mercado (e a comunidade open-source) de tecnologia evoluíram ao ponto da criação de frameworks e bibliotecas poderosas para facilitar o desenvolvimento destas UIs.

Muitos projetos contribuíram ou contribuem até hoje para esses processos, como: *jQuery*, *Backbone.js*, *Ember.js*, *Knockout.js*, *AngularJS*, *React*, *Angular*, *Vue.js e Svelte*.

Não é mentira a afirmação de que: *"Todo dia um novo framework/biblioteca JS é criado"*.

O pessoal se empolga muito!

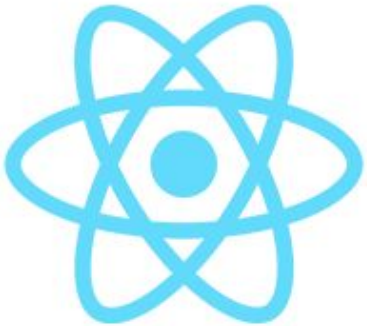
SPAx x Frameworks de Interação

Uma SPA competente deve estar bem resolvida (junto com sua stack de tecnologias) quanto a:

- Comunicação com o Servidor (AJAX + REST ou GraphQL)
- Gerenciamento de Rotas e Mudança de Tela
- Componentes Visuais Padronizados
- Templating (HTML e CSS)

Sempre escolha um framework (ou biblioteca) que pode de alguma forma cumprir estes requisitos

Frameworks JS mais Populares do Mercado 🚨



Star 204.9k

[React]

Star 87.2k

[Angular]

Star 203k

[Vue.js]

Star 66.6k

[Svelte]

Angular

Começaremos nossa jornada pelo mundo dos frameworks (e libs) JS através do Angular, um dos principais players da indústria atual.

- Criado **(2016)** e mantido pela Equipe Angular do Google e por uma comunidade de desenvolvedores open-source
- Caracterizado como um **framework** de aplicações web que se vende como "uma plataforma completa para criação de aplicações web e mobile."
- Não confundir com o finado AngularJS *(historinha triste)*
- Atualmente possui uma performance de interação tão boa quanto o React e o Vue.js
- Baseado na hierarquia de componentes como o seu principal conceito arquitetônico e possui muita lógica baseada em Templates
- Recomenda-se o uso de **TypeScript** (https://www.typescriptlang.org) como linguagem oficial de programação

Criando um projeto com Angular

A maioria dos frameworks relevantes do mercado possuem ferramentas iniciais ou indicadas para iniciar novos projetos.

E Angular também possui a sua ferramenta oficial de projetos: o **Angular CLI**.

Para instalar de forma universal em sua máquina utilize o comando em seu Terminal/Prompt:

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

Recomendo também dar uma olhada no link oficial de setup do projeto:
https://angular.io/guide/setup-local#install-the-angular-cli

Criando um projeto com Angular

Após instalado você poderá iniciar seu 1º projeto 100% baseado em Angular com o seguinte comando:

```
ng new nome-do-meu-projeto-angular
```

E para rodar seu novo projeto:

```
cd nome-do-meu-projeto-angular
ng serve --open
```

Componentes em Angular

A maioria dos frameworks de mercado trabalha com a ideia de que uma aplicação é um grande conjunto de pequenas peças. A maioria dessas peças (customizadas ou não) costumam ser chamadas de "***Componentes***".

Normalmente um componente Angular é baseado em 3 arquivos:

- **form-login.component.ts** => Lógica / Controlador do Component
- **form-login.component.html** => Template HTML
- **form-login.component.css** => Estilização

Componentes em Angular

É possível gerar um novo componente em sua aplicação através do seguinte comando:

```
ng generate component form-login
```

Componentes em Angular

form-login/form-login.component.ts

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

@Component({
  selector: 'app-form-login',
  templateUrl: './form-login.component.html',
  styleUrls: ['./form-login.component.css']
})
export class FormLoginComponent {
  title = "Título vindo do Controlador"
}
```

Componentes em Angular

form-login/form-login.component.html

```
<h1>{{title}}</h1>
<p>form-login works!</p>
```

Componentes em Angular

form-login/form-login.component.css

```css
h1 {
  font-size: 32px;
  color: red;
}
```

Componentes em Angular

Sobre Templates em Angular

As templates Angular combinam o HTML tradicional com um código de marcação próprio do Angular. Esse código de marcação Angular pode modificar os elementos HTML antes deles serem exibidos em tela. Usando a sintaxe de "binding markup" é possível conectar sua aplicação Angular ao DOM (JavaScript).

Existem dois tipos fundamentais de bindings em Angular:

- ***Event binding*** => permite seu aplicativo observar e responder a eventos de interação do usuário (onClick, onChange e etc)
- ***Property binding*** => permite ao aplicativo interpolar e exibir valores de variáveis computadas em seu HTML

Componentes em Angular

Sobre Templates em Angular

Antes de uma tela ser exibida, o Angular analisa as diretivas e transforma a sintaxe de ligação da template a fim de modificar os elementos HTML e o DOM, de acordo com os dados e a lógica de sua programação.

Diferente de outros frameworks/libs, o Angular suporta **ligação de dados bidirecional *(two-way data binding)***, o que significa que alterações no DOM, como opções e inputs do usuário, também são refletidas nos dados de sua aplicação.

Exemplo de template HTML + Angular

```
<div class="meu-app">
    <h4>{{meuValor}}</h4>
    <button [disabled]="meuValor === 2" (click)="atualizaValor()">
        Atualiza Valor
    </button>
</div>
```

Na template acima ***meuValor*** será uma variável baseada na parte lógica de nosso componente. E o evento de onclick definido através de ***(click)="atualizaValor()"*** irá acionar uma função chamada ***atualizaValor()*** também na camada lógica de nosso componente.

Vale a pena dar uma olhada na documentação completa de templates:
https://angular.io/guide/template-syntax

Templates em Angular: *Diretivas Estruturais*

Alguns elementos chave sobre as templates de Angular *vem de fábrica* com o projeto, são as **structural directives**.

Estas diretivas estruturais moldam ou reformulam a estrutura do DOM, geralmente adicionando, removendo e manipulando os elementos as quais estão conectadas.

As **structural directives** mais comuns são:

👉 ***NgIf*** => condicionalmente adiciona ou remove um elemento do DOM (ou da tela)

```
<div *ngIf="count > 1">Meu Contador: false</div>
```

👉 ***NgForOf*** => repete a template para cada item de uma lista

```
<ul>
    <li *ngFor="let item of items">{{item}}</li>
</ul>
```

NgModel

Two-way data binding para campos de formulário

Ao desenvolver formulários, você geralmente exibe em um campo uma variável de dados e atualiza essa propriedade quando o usuário faz alterações.

A ligação de dados bidirecional ***(Two-way data binding)*** com a diretiva ***ngModel*** deixa o "caminho" menos difícil:

```
<input [(ngModel)]="name" />
```

Sem o uso do ***ngModel*** teríamos que fazer isso daqui em todos os campos:

```
<input [value]="name" (input)="name=$event.target.value" />
```

NgModel

Two-way data binding para campos de formulário

Antes de começar a usar ***ngModel*** será necessário importar o módulo ***FormsModule*** e adicioná-lo a uma lista de módulos da aplicação com a diretiva ***NgModule***:

```
import { NgModule } from "@angular/core"; // <-- NgModule
import { FormsModule } from "@angular/forms"; // <-- FormsModule
...
@NgModule({
    // array com módulos "dependentes" para funcionamento da aplicação
    imports: [FormsModule],
})
export class AppModule {}
```

SO...
LET'S CODE!

obrigado 🚀