1 Introdução

O desenvolvimento moderno de aplicações Web é comumente realizado utilizando-se diferentes tecnologias. Muitas vezes, sua combinação dá origem a algo que tem levado o nome de desenvolvimento “Full Stack”. Uma solução desenvolvida segundo esse paradigma possui, em geral, duas aplicações independentes que se comunicam por meio de uma interface bem definida. Uma delas oferece interfaces gráfica para o usuário e geralmente é chamada de aplicação Front End. A outra é responsável por disponibilizar as funcionalidades do sistema e leva o nome de aplicação Back End. Em geral, ela faz uso de um sistema independente que possui implementações eficientes para operações de acesso à memória secundária.

Além disso, seja no Front End ou no Back End, é comum o uso de diferentes *frameworks* que supostamente entregam um nível maior de abstração e promovem a produtividade dos desenvolvedores.

Nos dias atuais, uma das combinações mais utilizadas para esse fim tem a sigla “**MEAN**” associada, a qual deriva de “**MongoDB**”, “**Express**”, “**Angular**” e “**NodeJS**”.

Neste material desenvolveremos uma aplicação que faz uso da “pilha” MEAN.

2 Desenvolvimento

2.1 (Entendendo a pilha MEAN) Os quatro itens que compõem a pilha MEAN são descritos a seguir.

MongoDB: Trata-se de um Sistema Gerenciador de Banco de Dados cujo modelo de dados é baseado em documentos. Ele se encaixa na classe de sistemas conhecida como **NoSQL**.

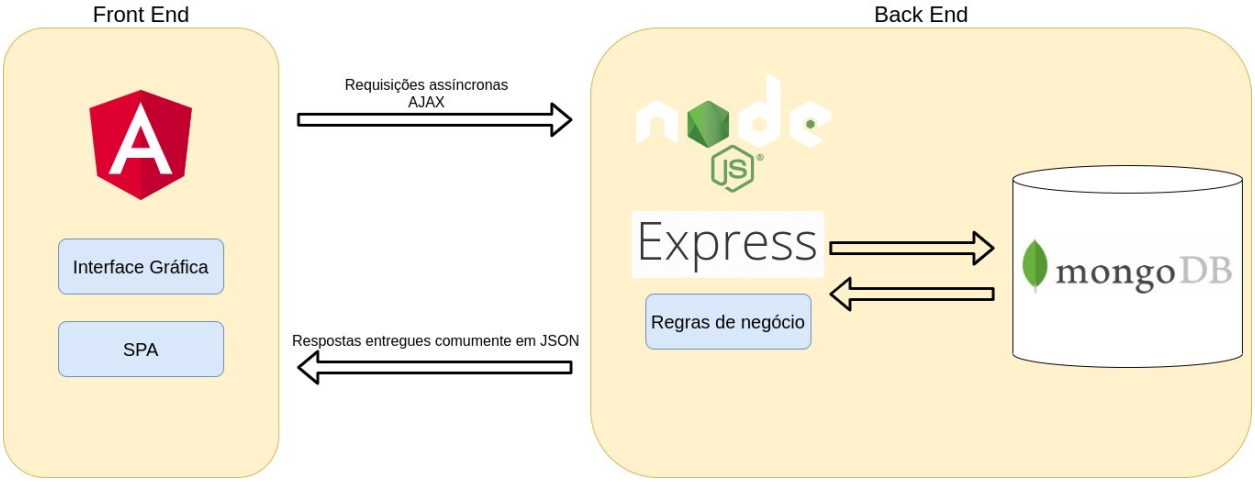
Express: O desenvolvimento de servidores que utilizam o protocolo HTTP pode ser um tanto trabalhoso. O Express é um framework próprio para NodeJS que visa simplificar o código para tratamento de requisições HTTP, evitando que tenhamos que escrever código repetido.

Angular: O Angular é um dos principais frameworks Front End nos dias atuais. Trata-se de uma solução que tem como um de seus principais objetivos simplificar o desenvolvimento de Single Page Applications.

NodeJS: É uma solução que viabiliza a escrita de código Javascript que executa do lado do servidor.

A Figura 2.1.1 ilustra a pilha MEAN.

Figura 2.1.1



2.2 (Criando o Projeto Front End) Escolha uma pasta no seu sistema de arquivos. Ela será seu workspace. Cada uma de suas subpastas representará um novo projeto. Abra um terminal vinculado a ela e digite o seguinte comando para criar um novo projeto.

ng new nome-do-projeto

Abra uma instância do VS Code vinculada à pasta recém criada com

code .

A seguir, coloque o servidor de testes em execução com

ng serve

Pode ser de interesse usar os terminais “embutidos” que o VS Code possui.

Nota: A extensão “**Angular Essentials”** (Autor **John Papa**) disponível no Marketplace do VS Code pode ser de interesse.

Nota: A extensão **Material Icon Theme** adiciona ícones da especificação Material Design ao VS Code. Caso opte por sua instalação, pode ser necessário clicar em File >> Preferences >> File Icon Theme para habilitá-la.

2.3 (Componente para a inserção de clientes) A aplicação que criaremos permitirá a realização de operações de persistência de dados de clientes. O primeiro componente Angular que criaremos permitirá a inserção de clientes.

- Clique com o direito na pasta **app** e crie uma subpasta chamada **clientes**. Dentro dela, crie uma subpasta chamada **cliente-inserir**.

- A seguir, crie um arquivo chamado **cliente-inserir.component.ts**.

- Crie também um arquivo chamado **cliente-inserir.component.html**.

Eles serão usados para definir um componente Angular que, como o nome sugere, permite a inserção de clientes. Veja o código inicial do componente na Listagem 2.3.1.

Listagem 2.3.1

```
import { Component } from '@angular/core';
@Component({
selector: 'app-cliente-inserir',
templateUrl: './cliente-inserir.component.html',
})
export class ClienteInserirComponent {}
```

- Adicione o código da Listagem 2.3.2 ao arquivo **cliente-inserir.component.html** afim de que possamos realizar um primeiro teste com o componente.

Listagem 2.3.2

```
<h2>Inserção de clientes</h2>
```

- A fim de utilizar o componente que acabamos de criar, abra o arquivo **app.component.html** e utilize o seu seletor,c omo na Listagem 2.3.3. O conteúdo inicial dele pode ser removido.

Listagem 2.3.3

```
<app-cliente-inserir></app-cliente-inserir>
```

Note que a página ficou em branco devido a um erro existente na aplicação. Ocorre que cada componente utilizado por ela deve fazer parte de um módulo. Precisamos dizer explicitamente a qual módulo pertence o componente ClienteInserirComponent. Para isso, abra o arquivo **app.module.ts** e adicione o seu nome ao vetor associado à chave **declarations**. Veja a Listagem 2.3.4.

Listagem 2.3.4

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

import { AppComponent } from './app.component';
import { ClienteInserirComponent } from './clientes/cliente-inserir/cliente-inserir.component';

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

- Clientes terão as propriedades **nome**, **fone** e **e-mail**. Assim, o template para inserção de clientes precisa exibir campos para a inserção desses valores, além de um botão que, quando clicado, coloque em execução um método que faça a inserção. Veja o código para o template na Listagem 2.3.5.

Listagem 2.3.5

```
<input type="text" placeholder="nome" />
<hr />
<input type="text" placeholder="fone" />
<hr />
<input type="text" placeholder="email" />
<hr />
<button>Inserir Cliente</button>
```

- Para disparar um método quando o botão for clicado, usaremos o **Event Binding** do Angular. Para isso, defina um método no arquivo **cliente-inserir.component.ts** como na Listagem 2.3.6.

Listagem 2.3.6

```
onAdicionarCliente() {
console.log('inserindo cliente...');
}
```

- A seguir, vincule-o ao evento **click** no botão, como mostra a Listagem 2.3.7.

Listagem 2.3.7

```
<input type="text" placeholder="nome" />
<hr />
<input type="text" placeholder="fone" />
<hr />
<input type="text" placeholder="email" />
<hr />
<button (click)="onAdicionarCliente()">Inserir Cliente</button>
```

- Clique no botão e verifique no console do seu navegador se o log está funcionando adequadamente.

- Podemos capturar os valores que o usuário digitar de formas diferentes. Aqui empregaremos o mecanismo conhecido como **Two way data binding**. Para isso, comece declarando, no componente, as três variávesi que a Listagem 2.3.8 exibe.

Listagem 2.3.8

```
export class ClienteInserirComponent {
nome: string;
fone: string;
email: string;
onAdicionarCliente() {
console.log('inserindo cliente...');
}
}
```

- A seguir, no template, use a diretiva **ngModel** em cada **input**. Veja a Listagem 2.3.9.

Listagem 2.3.9

```
<input type="text" placeholder="nome" [(ngModel)]="nome" />
<hr />
<input type="text" placeholder="fone" [(ngModel)]="fone" />
<hr />
<input type="text" placeholder="email" [(ngModel)]="email"/>
<hr />
<button (click)="onAdicionarCliente()">Inserir Cliente</button>
```

- Lembre-se de que a diretiva **ngModel** faz parte do módulo **FormsModule** que precisa ser importado por algum módulo de nossa aplicação. Como ela tem somente um, definido no arquivo **app.module.ts**, abra-o e importe o módulo FormsModule como na Listagem 2.3.10.

Listagem 2.3.10

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { ClienteInserirComponent } from './clientes/cliente-inserir.component';

@NgModule({
declarations: [AppComponent, ClienteInserirComponent],
imports: [BrowserModule, FormsModule],
providers: [],
bootstrap: [AppComponent],
})
export class AppModule {}
```

2.4 (Biblioteca Angular Material) Angular Material é uma biblioteca de componentes Angular. O aspecto visual de seus componentes segue as recomendações da especificação Material Design. Grande parte dos estilos de nossa aplicação será tratada pelos componentes dessa biblioteca. Sua página oficial pode ser visitada por meio do Link 2.4.1.

Link 2.4.1
https://material.angular.io/

- Para fazer a sua instalação, use o comando

ng add @angular/material

- Mantenha as opções de instalação com seu valor padrão (apenas aperte Enter).

2.5 (Estilizando o componente de inserção de clientes) Começaremos estilizando os elementos **input** do componente de inserção de clientes. Visite o Link 2.5.1 para ver alguns exemplos e começar a entender como a biblioteca funciona.

Link 2.5.1
https://material.angular.io/components/input/examples

- Desejamos utilizar a diretiva **matInput**. Para isso, vamos adicionar o módulo a que ela pertence ao módulo principal da aplicação, localizado no arquivo **app.module.ts**. Veja a Listagem 2.5.1.

Listagem 2.5.1

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { MatInputModule } from '@angular/material/input';

import { AppComponent } from './app.component';
import { ClienteInserirComponent } from './clientes/cliente-inserir.component';

@NgModule({
declarations: [AppComponent, ClienteInserirComponent],
imports: [
BrowserModule,
FormsModule,
BrowserAnimationsModule,
MatInputModule,
],
providers: [],
bootstrap: [AppComponent],
})
export class AppModule {}
```

- Como mostra a documentação, além de aplicar a diretiva **matInput**, precisamos englobar os elementos com o component **mat-form-field**. Veja a Listagem 2.5.2.

Listagem 2.5.2

```
<mat-form-field>
<input type="text" matInput placeholder="nome" [(ngModel)]="nome" />
</mat-form-field>
<hr />
<mat-form-field>
<input type="text" matInput placeholder="fone" [(ngModel)]="fone" />
</mat-form-field>
<hr />
<mat-form-field>
<input type="text" matInput placeholder="email" [(ngModel)]="email" />
</mat-form-field>
<hr />
<button (click)="onAdicionarCliente()">Inserir Cliente</button>
```

- É muito comum o uso de containeres de conteúdo estilizados como “cartões”. O Angular Material possui um componente assim, que utilizaremos agora. Para isso, precisamos importar o módulo a que ele pertence e, a seguir, utilizá-lo no template. Veja o ajuste a ser feito no arquivo **app.module.ts** na Listagem 2.5.3.

Listagem 2.5.3

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { MatInputModule } from '@angular/material/input';
import { MatCardModule } from '@angular/material/card';

import { AppComponent } from './app.component';
import { ClienteInserirComponent } from './clientes/cliente-inserir.component';

@NgModule({
declarations: [AppComponent, ClienteInserirComponent],
imports: [
BrowserModule,
FormsModule,
BrowserAnimationsModule,
MatInputModule,
MatCardModule,
],
providers: [],
bootstrap: [AppComponent],
})
export class AppModule {}
```

- O template deve ser ajustado como na Listagem 2.5.4.

Listagem 2.5.4

```
<mat-card>
<mat-form-field>
<input type="text" matInput placeholder="nome" [(ngModel)]="nome" />
</mat-form-field>
<hr />
<mat-form-field>
<input type="text" matInput placeholder="fone" [(ngModel)]="fone" />
</mat-form-field>
<hr />
<mat-form-field>
<input type="text" matInput placeholder="email" [(ngModel)]="email" />
</mat-form-field>
<hr />
<button (click)="onAdicionarCliente()">Inserir Cliente</button>
</mat-card>
```

- Além dos estilos já existentes nos componentes do Angular Material, também aplicaremos alguns próprios de nossa aplicação, usando CSS simples. Cada componente Angular pode ter uma coleção de arquivos .css própria. Vamos definir um arquivo chamado **cliente-inserir.component.css** na pasta **clientes** e, a seguir, referenciá-lo na propriedade **styleURLs** no componente. Veja a Listagem 2.5.5.

Listagem 2.5.5

```
import { Component } from '@angular/core';
@Component({
selector: 'app-cliente-inserir',
templateUrl: './cliente-inserir.component.html',
styleUrls: ['./cliente-inserir.component.css'],
})
export class ClienteInserirComponent {
```

- No arquivo CSS, escreva uma regra que tenha como alvo componentes do tipo **mat-card**. Ajuste sua largura e margem para que ele fique centralizado. Assim será possível ver os estilos do cartão. Veja a Listagem 2.5.6.

Listagem 2.5.6

```
mat-card {
width: 80%;
margin: auto;
}
```

- Além disso, ainda no arquivo CSS, escreva uma regra indicando que os elementos input devem tomar 100% de largura. Para que isso ocorra, contudo, seu pai direto deve também ter essa configuração. Veja a Listagem 2.5.7.

Listagem 2.5.7

```
mat-form-field,
input {
width: 100%;
}
```

- Caso queira, remova os elementos **hr** do template.

- O botão também precisa ser estilizado usando recursos do Angular Material. O procedimento é o mesmo: importamos o módulo apropriado e aplicamos a diretiva ao elemento no template. As listagens 2.5.8 e 2.5.9 mostram os ajustes dos arquivos app.module.ts e cliente-inserir.component.ts, respectivamente.

Listagem 2.5.8

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { MatInputModule } from '@angular/material/input';
import { MatCardModule } from '@angular/material/card';
import { MatButtonModule } from '@angular/material/button';

import { AppComponent } from './app.component';
import { ClienteInserirComponent } from './clientes/cliente-inserir.component';

@NgModule({
declarations: [AppComponent, ClienteInserirComponent],
imports: [
BrowserModule,
FormsModule,
BrowserAnimationsModule,
MatInputModule,
MatCardModule,
MatButtonModule,
],
providers: [],
bootstrap: [AppComponent],
})
export class AppModule {}
```

Listagem 2.5.9

```
<mat-card>
<mat-form-field>
<input type="text" matInput placeholder="nome" [(ngModel)]="nome" />
</mat-form-field>
<mat-form-field>
<input type="text" matInput placeholder="fone" [(ngModel)]="fone" />
</mat-form-field>
<mat-form-field>
<input type="text" matInput placeholder="email" [(ngModel)]="email" />
</mat-form-field>
<button color="primary" mat-raised-button (click)="onAdicionarCliente()">
Inserir Cliente
</button>
</mat-card>
```

2.6 (Cabeçalho) Vamos adicionar um novo componente Angular para representar o cabeçalho da aplicação.

- Para criá-lo, use

ng g c Cabecalho --skip-tests

- O Angular Material possui um componente próprio para barras desse tipo, presente no módulo MatToolbarModule. Importe-o no arquivo app.module.ts como mostra a Listagem 2.6.1.

Listagem 2.6.1

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { MatInputModule } from '@angular/material/input';
import { MatCardModule } from '@angular/material/card';
import { MatButtonModule } from '@angular/material/button';
import { MatToolbarModule } from '@angular/material/toolbar';

import { AppComponent } from './app.component';
import { ClienteInserirComponent } from './clientes/cliente-inserir.component';

@NgModule({
declarations: [AppComponent, ClienteInserirComponent],
imports: [
BrowserModule,
FormsModule,
BrowserAnimationsModule,
MatInputModule,
MatCardModule,
MatButtonModule,
MatToolbarModule
],
providers: [],
bootstrap: [AppComponent],
})
export class AppModule {}
```

- No arquivo **cabecalho.component.html**, adicione o conteúdo da Listagem 2.6.2.

Listagem 2.6.2

```
<mat-toolbar color="primary">Clientes</mat-toolbar>
```

- O novo componente pode ser utilizado por meio de seu seletor. Faça isso no arquivo **app.component.html**, como na Listagem 2.6.3.

Listagem 2.6.3

```
<app-cabecalho></app-cabecalho>
<app-cliente-inserir></app-cliente-inserir>
```

- Note que o cartão está colado no cabeçalho. Vamos ajustar isso especificando que o cartão faz parte da seção principal da página (com uma tag **main** do HTML5) e aplicar um valor para sua propriedade **margin-top**. As listagens 2.6.4 e 2.6.5 mostram os ajustes a serem feitos nos arquivos **cabecalho.component.html** e **app.component.css**, respectivamente.

Listagem 2.6.4

```
<app-cabecalho></app-cabecalho>
<main>
<app-cliente-inserir></app-cliente-inserir>
</main>
```

Listagem 2.6.5

```
main {
margin-top: 1rem;
}
```

2.7 (Componente para exibir a lista de clientes) Criaremos, agora, um componente cuja finalidade será exibir a lista de clientes (uma vez que ela exista).

- Para criá-lo, use o comando

ng g c clientes/cliente-lista --skip-ests

- Ajuste o arquivo **app.component.html** para utilizá-lo, como na Listagem 2.7.1.

Listagem 2.7.1

```
<app-cabecalho></app-cabecalho>
<main>
<app-cliente-inserir></app-cliente-inserir>
<app-cliente-lista></app-cliente-lista>
</main>
```

- O Angular Material possui um componente interessante para a exibição de listas. Ele se chama **Expansion Panel**. Como de costume, para utilizá-lo, precisamos importar o módulo a que ele pertence no módulo principal da aplicação, como mostra a Listagem 2.7.2.

Listagem 2.7.2

```
import { MatExpansionModule } from '@angular/material/expansion';

imports: [
BrowserModule,
FormsModule,
BrowserAnimationsModule,
MatInputModule,
MatCardModule,
MatButtonModule,
MatToolbarModule,
MatExpansionModule,
],
```

- Segundo a documentação oficial, seu uso básico consiste em um componente principal chamado **mat-accordion** que engloba um **mat-expansion-panel**. Por sua vez, um mat-expansion-panel pode ter um **mat-expansion-panel-header** (que pode armazenar o título da lista, por exemplo) e algum conteúdo simples logo abaixo. Veja a Listagem 2.7.3. Use seu conteúdo no arquivo **cliente-lista.component.html**.

Listagem 2.7.3

```
<mat-accordion>
<mat-expansion-panel>
<mat-expansion-panel-header>Lista de clientes</mat-expansion-panel-header>
<p>Conteúdo</p>
<p>Conteúdo</p>
</mat-expansion-panel>
</mat-accordion>
```

- Note que a largura dele precisa ser ajustada para ser condizente com a largura do cartão. Para isso, comece **removendo** o estilo do mat-card existente no arquivo **cliente-inserir.component.css**. A seguir, adicione aplique as mesmas configurações, exibidas na Listagem 2.7.4, ao elemento **main** do componente principal da aplicação, no arquivo **app.component.css**.

Listagem 2.7.4

```
main {
margin-top: 1rem;
width: 80%;
margin-left: auto;
margin-right: auto;
}
```

- A seguir, vamos adicionar uma regra CSS ao arquivo **cliente-lista.component.css** para especificar um valor de margin-top, a fim de descolar a lista do cartão. Veja a Listagem 2.7.5.

Listagem 2.7.5

```
:host {
margin-top: 1rem;
display: block;
}
```

- No momento, a lista exibe somente dois parágrafos como conteúdo. Vamos definir um vetor de objetos JSON que ela se encarregará de exibir. Adicione o vetor da Listagem 2.7.6 ao arquivo **cliente-inserir.component.ts**.

Listagem 2.7.6

```
import { Component, OnInit } from '@angular/core';
@Component({
selector: 'app-cliente-lista',
templateUrl: './cliente-lista.component.html',
styleUrls: ['./cliente-lista.component.css'],
})
export class ClienteListaComponent implements OnInit {
clientes = [
{
nome: 'José',
fone: '11223344',
email: 'jose@email.com',
},
{
nome: 'Maria',
fone: '22334455',
email: 'maria@email.com',
},
];
constructor() {}

ngOnInit(): void {}
}
```

- A fim de exibir os elementos existentes na lista, utilizaremos a diretiva estrutura ***ngFor**. Cada elemento da lista será exibido em um **mat-expansion-panel**. Veja a Listagem 2.7.7.

Listagem 2.7.7

```
<mat-accordion>
<mat-expansion-panel *ngFor="let cliente of clientes">
<mat-expansion-panel-header
>Nome: {{ cliente.nome }}</mat-expansion-panel-header
>
<p>Fone: {{ cliente.fone }}</p>
<hr/>
<p>Email: {{ cliente.email }}</p>
</mat-expansion-panel>
</mat-accordion>
```

- Vamos garantir que a lista somente será renderizada caso exista pelo menos um cliente nela Além disso, caso ela esteja vazia, vamos exibir um texto informando isso. Para renderizar elementos de forma condicional, usamos a diretiva estrutural ***ngIf**. Veja a Listagem 2.7.8.

Listagem 2.7.8

```
<mat-accordion *ngIf="clientes.length > 0">
<mat-expansion-panel *ngFor="let cliente of clientes">
<mat-expansion-panel-header
>Nome: {{ cliente.nome }}</mat-expansion-panel-header
>
<p>Fone: {{ cliente.fone }}</p>
<hr />
<p>Email: {{ cliente.email }}</p>
</mat-expansion-panel>
</mat-accordion>
<p class="mat-body-1" style="text-align: center;" *ngIf="clientes.length <= 0">
Nenhum cliente cadastrado
</p>
```

- Comente a definição da lista de clientes que contém dados fixos e crie uma lista vazia para testar, como na Listagem 2.7.9.

Listagem 2.7.9

```
// clientes = [
// {
// nome: 'José',
// fone: '11223344',
// email: 'jose@email.com',
// },
// {
// nome: 'Maria',
// fone: '22334455',
// email: 'maria@email.com',
// },
// ];

clientes = [];
```

2.8 (Lidando com o cadastro de clientes) O componente cliente-inserir se encarrega de capturar os dados digitados pelo usuário. O botão de inserção de clientes também faz parte dele. Quando o botão for clicado, ele precisa construir um objeto cliente e informar ao componente que o utiliza que o evento de inserção de cliente aconteceu.

- Vamos começar ajustando o componente cliente-inserir. Ele usará um **EventEmitter** para informar sobre o evento de inserção de cliente. Veja a Listagem 2.8.1.

Listagem 2.8.1

```
import { Component, EventEmitter, Output } from '@angular/core';
@Component({
selector: 'app-cliente-inserir',
templateUrl: './cliente-inserir.component.html',
styleUrls: ['./cliente-inserir.component.css'],
})
export class ClienteInserirComponent {
@Output() clienteAdicionado = new EventEmitter();
nome: string;
fone: string;
email: string;
onAdicionarCliente() {
const cliente = {
nome: this.nome,
fone: this.fone,
email: this.email,
};
this.clienteAdicionado.emit(cliente);
}
}
```

- Agora podemos usar o **event binding** no componente principal para especificar uma função que deve ser executada assim que esse evento (cliente adicionado) acontecer. Veja a Listagem 2.8.2.

Listagem 2.8.2

```
<app-cabecalho></app-cabecalho>
<main>
<app-cliente-inserir
(clienteAdicionado)="onClienteAdicionado($event)"
></app-cliente-inserir>
<app-cliente-lista></app-cliente-lista>
</main>
```

- A função onClienteAdicionado ainda não existe. Cabe ao componente principal defini-la. Faça a sua implementação inicial como mostra a Listagem 2.8.3 e faça uma inserção para ver se tudo está funcionando.

Listagem 2.8.3

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

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css'],
})
export class AppComponent {
onClienteAdicionado(cliente) {
console.log(cliente);
}
}
```

- Ajuste a função onClienteAdicionado para que ela adicione o cliente recebido a uma lista. A lista precisa ser definida pelo componente principal. Ela será entregue ao componente responsável pela exibição dos clientes em breve. Veja a Listagem 2.8.4.

Listagem 2.8.4

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

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css'],
})
export class AppComponent {
clientes = [];
onClienteAdicionado(cliente) {
this.clientes = [...this.clientes, cliente];
}
}
```

- O componente principal da aplicação precisa entregar a lista de clientes para o componente responsável por exibi-la. Embora ele já possua uma lista, ela ainda não é uma propriedade com a qual se possa estabelecer vínculo. Para tornar isso possível, usamos a anotação **Input**. Veja o ajuste a ser feito no arquivo **cliente-lista.component.ts** na Listagem 2.8.5.

Listagem 2.8.5

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

@Component({
selector: 'app-cliente-lista',
templateUrl: './cliente-lista.component.html',
styleUrls: ['./cliente-lista.component.css'],
})
export class ClienteListaComponent implements OnInit {
@Input() clientes = [];

constructor() {}

ngOnInit(): void {}
}
```

- Agora podemos fazer o **property binding**, como na Listagem 2.8.6.

Listagem 2.8.6

```
<app-cabecalho></app-cabecalho>
<main>
<app-cliente-inserir
(clienteAdicionado)="onClienteAdicionado($event)"
></app-cliente-inserir>
<app-cliente-lista [clientes]="clientes"></app-cliente-lista>
</main>
```

2.9 (Definindo o que é um cliente explicitamente) É sempre uma boa prática de programação especificar o máximo de informação que pudermos em tempo de compilação. Em particular, temos utilizado a estrutura que representa clientes sem dizer explicitamente quais propriedades ela tem. A lista de clientes, neste momento, pode armazenar qualquer tipo de dado. Vamos especificar o tipo cliente e utilizá-lo nos componentes da aplicação.

- Para especificar o novo tipo, crie um novo arquivo na pasta **clientes** chamado **cliente.model.ts**. Veja seu conteúdo na Listagem 2.9.1.

Listagem 2.9.1

```
export interface Cliente {
nome: string;
fone: string;
email: string;
}
```

- No arquivo **app.component.ts**, utilize o novo contrato para definir a lista, como na Listagem 2.9.2.

Listagem 2.9.2

```
import { Component } from '@angular/core';
import { Cliente } from './clientes/cliente.model';

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css'],
})
export class AppComponent {
clientes: Cliente[] = [];
onClienteAdicionado(cliente) {
this.clientes = [...this.clientes, cliente];
}
}
```

- Faça o mesmo no arquivo **cliente-lista.component.ts**, como na Listagem 2.9.3.

Listagem 2.9.3

```
import { Component, OnInit, Input } from '@angular/core';
import { Cliente } from '../cliente.model';
@Component({
selector: 'app-cliente-lista',
templateUrl: './cliente-lista.component.html',
styleUrls: ['./cliente-lista.component.css'],
})
export class ClienteListaComponent implements OnInit {
@Input() clientes: Cliente[] = [];

constructor() {}

ngOnInit(): void {}
}
```

- Faça o mesmo também no arquivo **cliente-inserir.component.ts**. Veja a Listagem 2.9.4.

Listagem 2.9.4

```
import { Component, EventEmitter, Output } from '@angular/core';
import { Cliente } from '../cliente.model';
@Component({
selector: 'app-cliente-inserir',
templateUrl: './cliente-inserir.component.html',
styleUrls: ['./cliente-inserir.component.css'],
})
export class ClienteInserirComponent {
@Output() clienteAdicionado = new EventEmitter<Cliente>();
nome: string;
fone: string;
email: string;
onAdicionarCliente() {
const cliente: Cliente = {
nome: this.nome,
fone: this.fone,
email: this.email,
};
this.clienteAdicionado.emit(cliente);
}
}
```

Referências

Angular. 2020. Disponível em <https://angular.io>. Acesso em agosto de 2020.

Angular Material UI component library. 2020. Disponível em <https://material.angular.io>. Acesso em agosto de 2020

Express - Node.js web application framework. 2020. Disponível em <https://expressjs.com>. Acesso em agosto de 2020.

Node.js. 2020. Disponível em <https://nodejs.org/en/>. Acesso em agosto de 2020.

The most popular database for modern apps | MongoDB. 2020. Disponível em <https://www.mongodb.com>. Acesso em agosto de 2020.