**CONFIGURAÇÃO**

O desafio foi desenvolvido com framework Angular. Para inicializá-lo localmente, é necessário instalação no Node.JS LTS (<https://nodejs.org/en/download>) e Angular conforme orientações na documentação disponível em <https://angular.io/guide/setup-local>.

**RECURSOS EXTERNOS**

Como framework de componentes, foi aplicado o uso do Angular Material (https://material.angular.io/). Para auxiliar na disposição de elementos no layout, foi aplicado o framework PrimeFlex (<https://primeflex.org/>). A biblioteca Lodash (<https://lodash.com/>) foi utilizada para manipulação de arrays. Além disso, fora empregados alguns recursos visuais da marca CAIXA, tais como cores e tipografia.

**ESTRUTURA E FUNCIONAMENTO**

O projeto tem por objetivo exibição intuitiva de dados disponibilizados por uma API de imagens fictícias (<https://jsonplaceholder.typicode.com/photos/>). Para iniciar, isolei em uma classe de serviços a funcionalidade que consome dados da API, conforme recomendado como boa prática de programação na documentação do framework.

Foi utilizado para implementação a versão 17 do Angular. A versão traz como novidade uma sintaxe mais simplificada, dispensando a necessidade separação por módulos e trazendo o conceito de “*standalone components”*. Este recurso permite que os componentes “chamem” suas dependências a partir da propriedade ‘*imports’* do decorador “@*Component*”. Além de fluidez na escrita de código, os *bundles* ficaram mais leves, mantendo-se preservada possibilidade de separação de utilização de recursos por demanda, implementada através de “*lazy-load*” por meio de arquivos de rotas.

Como a aplicação proposta no desafio é muito pequena, não se fez necessária a criação de arquivo de rotas além daquela que já vem criada no projeto padrão. Para facilitar a manipulação de dados, foi criada uma interface para tipar por toda a aplicação os objetos retornados pela API. A interface está armazenada na pasta models sob título *Photo*. Para auxiliar na funcionalidade de filtragem das imagens, foi empregado o recurso nativo do framework conhecido como *Pipe*. Semelhante a uma ‘esteira’ do mundo real, um *pipe* permite um encadeamento de chamadas sequenciais, onde cada classe com este decorador armazena sua própria lógica de tratamento dos dados de entrada. Outra novidade nesta versão do framework é a nova sintaxe das diretivas de controle e renderização condicional. O \*ngIf ou \*ngFor foi substituído por uma forma escrita muito parecida com linguagens de programação tradicionais (@if(codição){bloco de execução} ou @for(let item of itens){ bloco de execução}).

A estrutura do código conta com apenas 4 componentes:

***AppComponent***: componente raíz criado pelo próprio CLI da aplicação. Foi reaproveitado para controlar os estados básicos da aplicação e armazenar os dados consumidos da API. Este componente abriga o fluxo principal da aplicação. Seguindo o *lifecycle* básico de componentes, é feita uma chamada assíncrona para API de fotos através do método *GetAll* do *PhotoService*. Enquanto aguarda essa resposta, é renderizado ao usuário um *loader* com a mensagem de ‘Carregando Imagens’ até que o processo seja concluído. Se a chamada concluir com sucesso, a variável Photos é inicializada com a resposta da API e os dados serão renderizados por componentes filho. Em caso de erro, a variável de estado ‘erroApi’ é setada como verdadeira, renderizando uma mensagem de falha ao usuário, dando a ele a opção de tentar carregar novamente. Este componente também abriga uma variável *string* utilizada para filtragem do *array* de imagens pelo título, além de uma variável booleana que serve como *switch* para dois modos de exibição distintos: tabela (lista) ou cards. Este componente também gerencia a visualização da imagem em tela cheia: esta funcionalidade foi implementada com comunicação entre componentes.

***VisualizarTabelaComponent***: Este componente encapsula os recursos para visualização dos dados através de uma matriz ou tabela. Os dados das imagens são recebidos por meio de *data-binding* e armazenados na variável *Photos*. Outra variável recebida do componente pai é a *inputSearchString*, que armazena o texto imputado pelo usuário e utilizado para filtragem de fotos pelo título. Algumas variáveis extras de controle foram empregadas para implementação de ordenação, que pode ser acessada pelos *labels* da tabela renderizada. A tabela foi configurada para exibir inicialmente 10 registros por vez, podendo o usuário listar até 100 registros através do seletor na parte inferior da matriz. Ao clicar na imagem minimizada, o componente envia para o componente pai os dados da foto clicada através de *EventEmitter*.

***VisualizarCardsComponent***: Este componente armazena recursos para exibição das imagens com layout de Cards. Os cards são uma maneira de exibição um pouco aumentada das imagens listadas, proporcionando ao usuário uma experiência de visualização mais detalhada da foto do que na opção de exibição por lista. Por uma questão de performance, são listadas apenas 40 fotos por vez. Ao chegar ao fim da lista, é exibido um botão dando opção ao usuário de renderizar mais 40 registros. Assim como no componente de lista, este componente também utiliza um emissor de eventos para enviar ao componente pai dados da foto clicada para exibição em tele cheia.

***VisualizarItemComponent***: Este componente armazena apenas uma pequena lógica para exibição da foto em maior resolução dentro de um modal.