SUMÁRIO

1. INTRODUÇÃO AO REACT

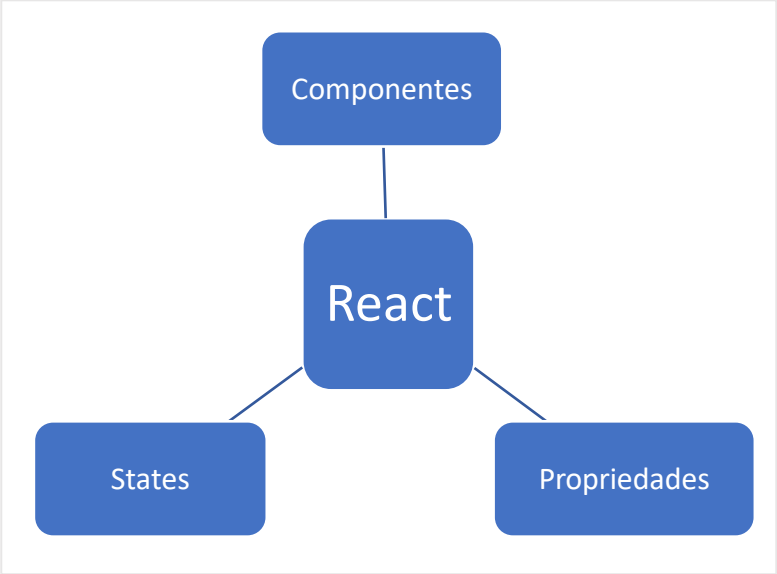
React é uma biblioteca JavaScript para construção de interfaces de usuário reativas e dinâmicas, que utiliza, de forma embutida, o Webpack, e uma completa "orientação a componentes". A documentação, para entender a tecnologia, pode ser encontrada aqui.

Em termos gerais, é possível e certeiro afirmar o seguinte: **react é composto por uma tríade de componentes, propriedades e states**.

Sobre o primeiro, e o que será reforçado diversas vezes ao longo deste documento, **componentes são** funções que retornam conteúdo em JSX. E o que é isso? Se trata de uma linguagem própria do React que une Javascript com marcações baseadas em HTML.

Sobre o segundo recurso, **as propriedades**, elas são o que no HTML original chamamos de atributos, e no contexto do JSX servem para transitar valores entre componentes.

Sobre o terceiro e último recurso que caracteriza a tecnologia, **os states são** dados, valores presentes, que servem de referencial para a ocorrência de eventos, ou para novas renderizações. É o que dá a "vida" para o React. Por exemplo, se o estado de X é 1, e se altera para 2, este novo estado se torna o presente de X, e desencadeia outros eventos que ao seu estado estavam atrelados.



1.1. INICIANDO O APP REACT

Acessando o local do projeto, via promp de comando, e tendo em mãos o NodeJs e um gerenciador de pacotes, como NPM ou Yarn, basta digitar "**npx create-react-app nome_app**", ou "**yarn create react-app nome_app"**, sendo "nome_app" o nome do seu projeto, ou aplicativo.

Após isso, e sendo um processo relativamente lento, os recursos começarão a ser baixados. Quando o processo for concluído, o aplicativo React estará pronto para ser desenvolvido.

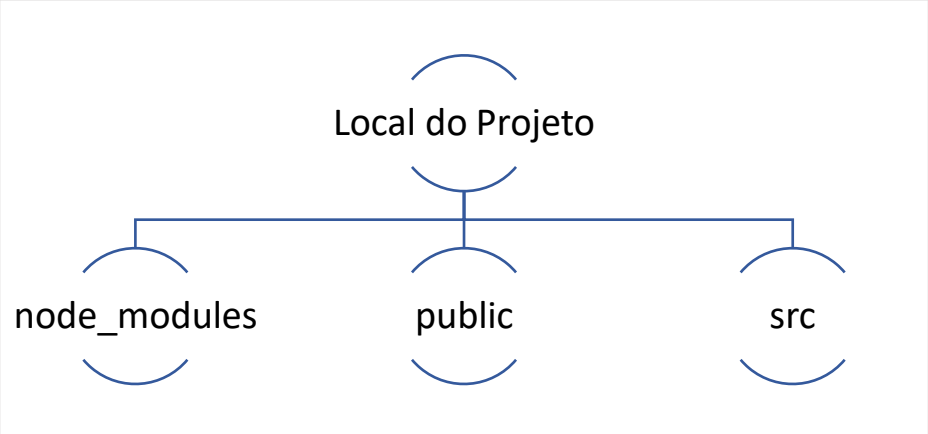
1.2. EXCLUSÃO DOS ARQUIVOS DESNECESSÁRIOS

O download do React traz consigo um modelo de interface pronto, e devido a isso uma série de arquivos desnecessários e que devem ser removidos.

Na pasta public pode ser mantido apenas o manifest.json e o index.html, **e na pasta src** apenas o index.js, index.css, o App.js e o App.css. Todos os outros, exceto package.json, gitignore, e o README, podem ser removidos.

1.3. ESTRUTURA DE DIRETÓRIOS

Após a conclusão do processo de download do React, uma série de pastas e arquivos poderão ser encontrados no local do projeto. Embora pareça uma árvore complexa de arquivos, como mostrado abaixo, são apenas alguns poucos que são estritamente necessários e que de fato serão utilizados com frequência.

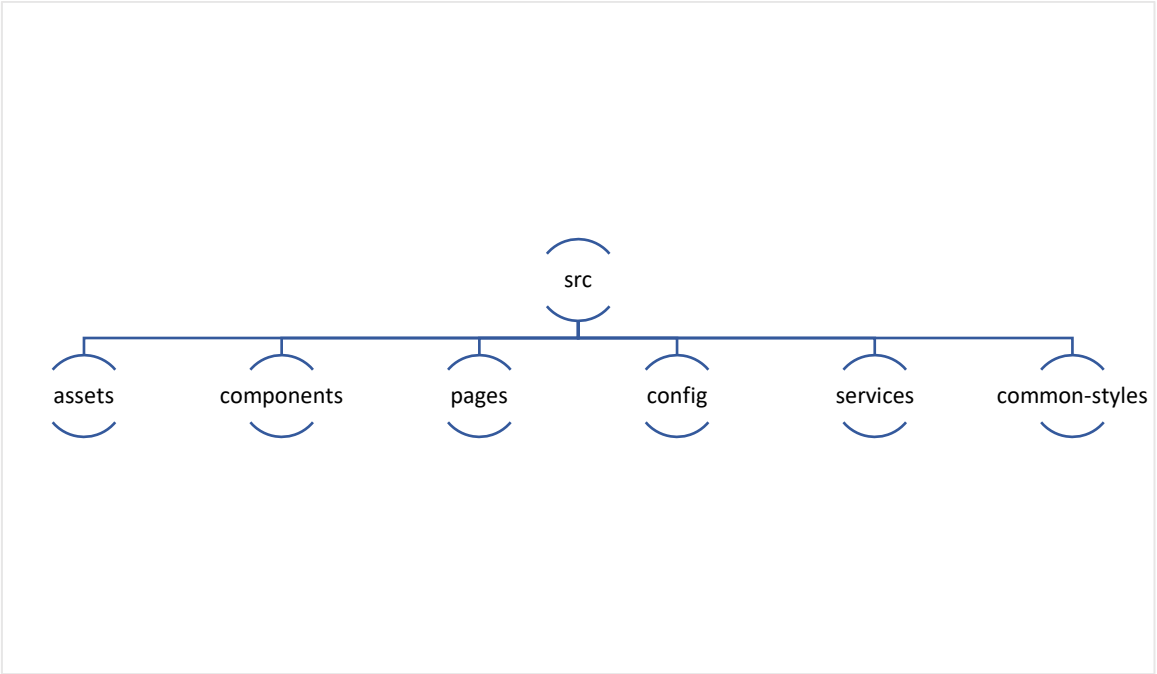


Pois então: na **pasta node_modules** estarão as dependências do projeto, gerenciadas pelo NPM; na **pasta public**, como na documentação, existirá o index.html, o manifest.json, e, talvez, alguns outros recursos se necessário para serem utilizados diretamente no index, sem o

processamento do Webpack; por último, na **pasta src** existirá tudo aquilo que molda a aplicação em si; a programação, os componentes, as folhas de estilo, as mídias, etc.

1.4. ORGANIZAÇÃO DA PASTA SRC

Adicionalmente, vale escrever sobre uma organização possível da pasta src, e digo possível porque a disposição dos arquivos e pastas é uma customização pessoal. Existem guias, mas não um padrão para isso.



A pasta assets irá conter todos os recursos como logos, imagens, vídeos, entre outras coisas do gênero; **a pasta components** terá componentes estruturais da aplicação, alocados em pastas próprias, como "/button", "/menu", "formulary", etc; **a pasta pages** irá conter as páginas da aplicação, e que também serão alocadas em pastas próprias; **a pasta config** terá o arquivo de rotas "routes.js"; **a pasta services** terá componentes de APIS utilizadas, e também outros recursos, como bibliotecas; **a pasta common-styles**, ou apenas styles, terá os componentes de estilo da aplicação que serão utilizados com frequência, isto é, reaproveitados ao longo da aplicação.

Ainda sobre os estilos, aqueles que forem específicos, para um fim ou outro, poderão existir na própria pasta do componente a que irão servir.

1.4.1. ORGANIZAÇÃO DA PASTA COMPONENTS

Na pasta components irão existir os componentes estruturais, e enfatizo essa classificação porque muito do que está fora desta pasta também é componente. Assim, se essa

pasta fosse realmente alocar todos os componentes, mais da metade da aplicação existiria dentro dela.

Pois bem: **componentes estruturais são** os blocos estruturais que irão compor a aplicação e as suas páginas, e que irão ser importados ao longo dela, em mais de um segmento. São, por exemplo, os botões, os formulários, as listas, as tabelas, também podem ser os cabeçalhos, os rodapés, enfim, tudo aquilo que pode ser classificado como algo que faz parte de um conjunto maior.

1.5. SOBRE O INDEX.JS E O INDEX.CSS

Há uma confusão comum entre o index.js e o App.js, assim como entre o index.css e o App.css, porque o App.js é invocado no index.js, mas poderia ser diretamente construído nele. E se assim fosse, adicionalmente, ao invés de um estilo para um e para o outro, haveria apenas para o index.js, o que no final das contas resultaria em uma redução de quatro arquivos para dois.

Essa junção do App.js com o index.js é possível, mas a separação é importante pois, como todo o resto da aplicação, a segmentação em componentes é útil para que a aplicação se torne reaproveitável e facilmente compreendida como um fluxo de partes acopladas.

Primeiro, **o index.js deve existir porque é o bunddle da aplicação**, isto é, o *Entry Point* do Webpack que opera implicitamente no React. Nele os recursos básicos do React são importados, como o próprio ‘react’ e o ‘react-DOM’. Além disso, é nele que ocorre a renderização do conteúdo que existirá e será gerenciado na raiz estrutural do programa, a div “root”. Já a sua folha de estilização, o index.css, pode ser entendido, por ser o index.js o *Entry Point*, como uma estilização global da aplicação – o “reset.css”.

Segundo, é interessante que exista o App.js porque ele é o componente que agrega todos os outros em si, formando o conteúdo da raiz da aplicação. Basta que ele seja importado, no index.js, para que ocorra a renderização do todo criado e agregado.

Em nítido contraste, se ele – o App.js - existisse no próprio index.js, ao invés de ser importado nele, junto a outros poucos recursos básicos, todas as importações para agregação de componentes também deveriam ocorrer no index.js.