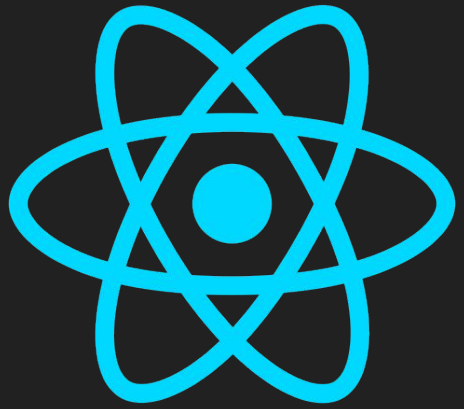
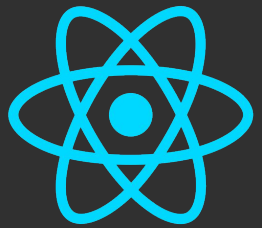


Curso de React

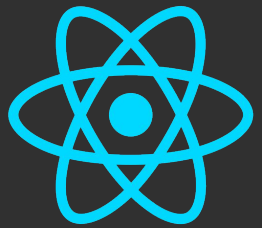
Aprenda do básico ao avançado

O que é React?

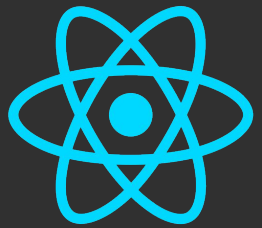
- React é uma **biblioteca JavaScript** para desenvolvimento de aplicações front-end;
- Estas aplicações são chamadas de **SPA** (Single Page Application);
- A arquitetura do React é baseada em **componentes**;
- Pode ser **inserido em um aplicação** ou podemos criar a aplicação apenas com React;
- É mantido pelo **Facebook**;



Instalando o Node.js

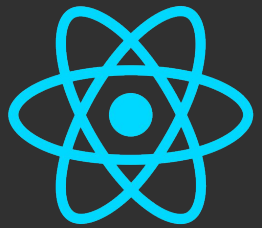


Instalando o VS Code



Hello World em React

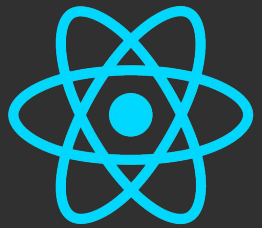
- Para criar as nossas aplicações em React vamos utilizar um executor de scripts do Node, que é o **npx**;
- Com o comando: **npx create-react-app <nome>** temos uma nova aplicação sendo gerada;
- Podemos iniciar a aplicação com **npm start**;
- Vamos ver na prática!



ATENÇÃO PARA A NOTA:

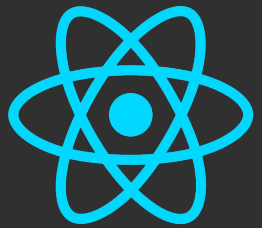
Estrutura base

- Há algumas pastas e arquivos chave para o desenvolvimento em React;
- **node_modules**: Onde as dependências do projeto ficam;
- **public**: Assets estáticos e HTML de inicialização;
- **src**: Onde vamos programar as nossas apps;
- **src/index.js**: Arquivo de inicialização do React;
- **src/App.js**: Componente principal da aplicação;



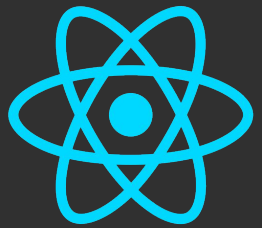
Extensão para React

- Há uma extensão no **marketplace** do VS Code que ajuda muito a programar em React;
- O nome dela é: **ES7 + React/Redux/React-Native** snippets;
- Conseguimos criar muito código com apenas alguns atalhos;
- Vamos instalá-la!



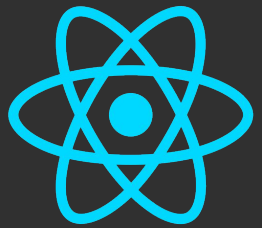
Preparando o Emmet para React

- **Emmet** é uma extensão nativa do VS Code que ajuda a escrever HTML mais rápido;
- Porém ela **não vem configurada** para o React!
- Temos que acessar **File > Settings > Extensions** e procurar por Emmet;
- Lá vamos incluir a linguagem: **javascript - javascriptreact**;
- Vamos configurar!



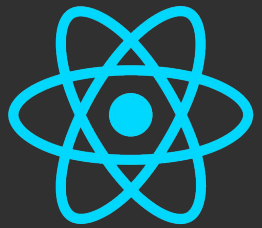
Como tirar melhor proveito do curso

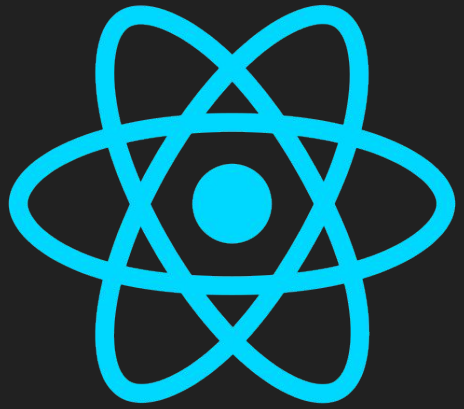
- **Sempre programe** os códigos das aulas!
- **O curso foi planejado sequencialmente**, se não domina o React não pule aulas, especialmente para os projetos;
- Faça todos os **exercícios** propostos;
- Tira as suas dúvidas no fórum (**Q & A**) e responda as dúvidas dos outros alunos que souber;
- Dica extra: assistir primeiro, executar depois;



Desafio 1

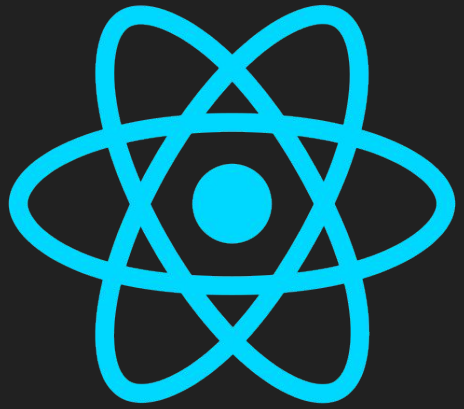
1. Entre no nosso arquivo App.js e adicione mais algum elemento HTML da sua escolha;
2. Crie uma regra de estilos em App.css que altere a cor do seu elemento;
3. Vá até o arquivo index.html e altere o título da aplicação na meta tag;





Introdução

Conclusão da seção

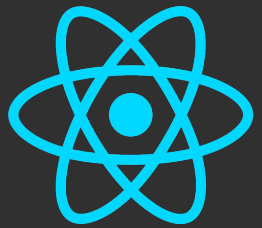


Fundamentos do React

Introdução da seção

Criando componentes

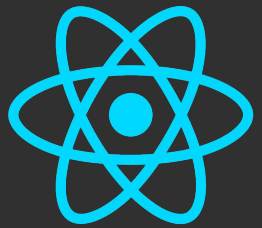
- Na maioria dos projetos os componentes ficam em uma pasta chamada **components**, que devemos criar;
- Geralmente são nomeados com a **camel case**: FirstComponent.js;
- No arquivo **criamos uma função**, que contém o código deste componente (a lógica e o template);
- E também precisamos **exportar esta função**, para reutilizá-lo;
- Vamos ver na prática!



ATENÇÃO PARA AS NOTAS:

Importando componente

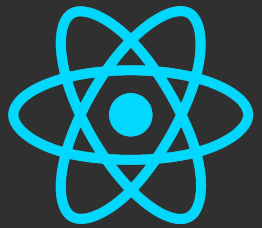
- A importação é a maneira que temos de **reutilizar o componente**;
- Utilizamos a sintaxe: **import X from ‘./componentes/X’** onde X é o nome do componente;
- Para colocar o componente importado em outro componente, precisamos colocá-lo em forma de tag: **<FirstComponent />**
- E então finalizamos o ciclo de importação;
- Vamos importar o FirstComponent em App;



ATENÇÃO PARA AS NOTAS:

JSX

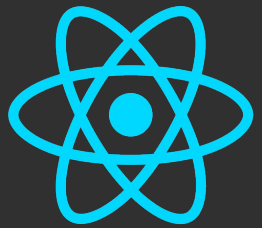
- **JSX** é o HTML do React;
- Onde vamos declarar as tags de HTML que serão exibidas no navegador;
- Ficam no **return** do componente;
- Temos algumas diferenças do HTML, por exemplo: class será **className**;
- Isso se dá pelas **instruções semelhantes de JS e HTML**, pois o JSX é JavaScript, então algumas terão nomes diferentes;
- O JSX pode ter apenas **um elemento pai**;



ATENÇÃO PARA A NOTA:

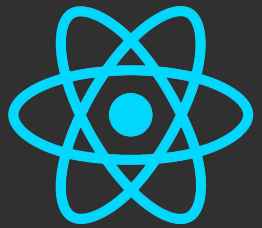
Comentários no componente

- Podemos inserir comentários de **duas maneiras** no componente;
- Na parte da função, onde é executada a lógica, a sintaxe é: **// Algum comentário**;
- E também no JSX: **{ /* Algum comentário */ }**
- As chaves nos permitem **executar sentenças em JavaScript**, veremos isso mais adiante;
- Vamos testar os comentários!



Template Expressions

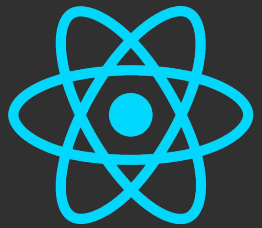
- **Template Expressions** é o recurso que nos permite executar JS no JSX e também **interpolar variáveis**;
- Isso será muito útil ao longo dos seus projetos em React;
- A sintaxe é: **{ algumCódigoEmJS }**
- **Tudo que está entre chaves é processado em JavaScript** e nos retorna um resultado;
- Vamos ver na prática!



ATENÇÃO PARA A NOTA:

Hierarquia de componentes

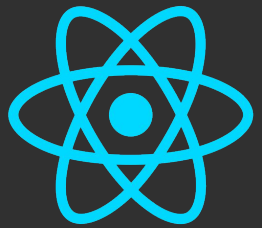
- Os componentes que criamos **podem ser reutilizados em vários componentes**;
- E ainda componentes **podem formar uma hierarquia**, sendo importados uns dentro dos outros, como fizemos em App;
- Vamos ver na prática estes conceitos!



ATENÇÃO PARA A NOTA:

Evento de click

- Os eventos para o front-end são **essenciais**;
- Em várias situações vamos precisar do click, como ao **enviar formulários**;
- No React os eventos já estão ‘prontos’, podemos utilizar **onClick** para ativar uma função ao clicar em um elemento;
- Esta função é criada na própria função do componente;
- As funções geralmente tem o padrão **handleAlgumaCoisa**;

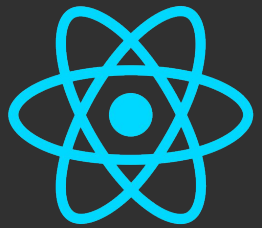


ATENÇÃO PARA A NOTA:

Funções no evento

- Quando as funções são simples, podemos **realizar a lógica no próprio evento**;
- Isso **torna nossa código mais 'complicado'**, por atrelar lógica com HTML;
- Mas em **algumas situações** é aplicável;
- Vamos ver na prática!

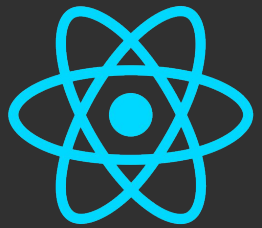
ATENÇÃO PARA A NOTA:



Funções de renderização

- Podemos criar **funções que retornam JSX**;
- Isso serve para criar situações que **dependam de outras condições**;
- Ou seja, o JSX a ser renderizado pode variar por alguma variável, por exemplo;
- Vamos ver na prática!

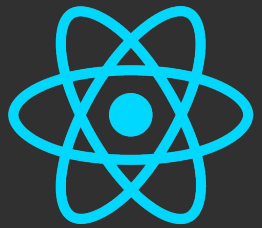
ATENÇÃO PARA A NOTA:

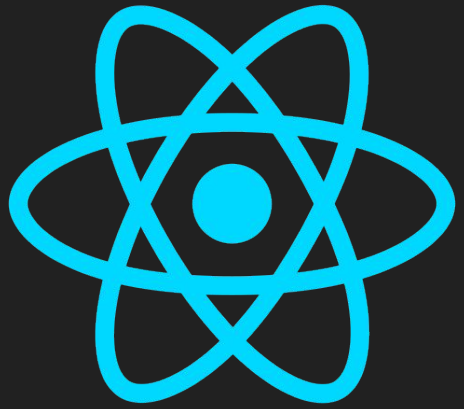


Desafio 2

1. Crie um componente chamado Challenge;
2. Importe-o em App.js;
3. No componente criado faça a criação de dois valores numéricos;
4. Imprima este valores no componente;
5. Crie também um evento de click que soma estes dois valores e exibe no console;

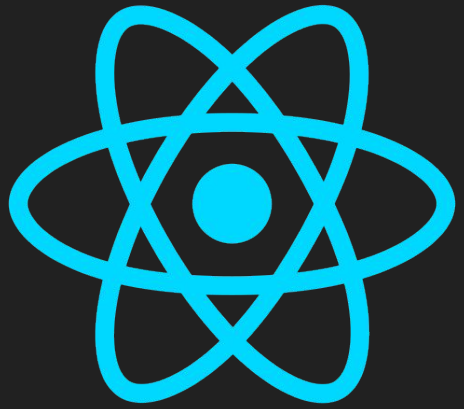
Resolução:





Fundamentos do React

Conclusão

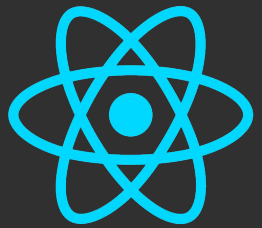


Avançando no React

Introdução da seção

Desafio 3

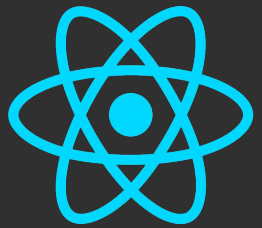
1. Crie um projeto para a nova seção;
2. Limpe o arquivo do componente principal;
3. E por fim coloque o título de Seção 3;



Imagens no React

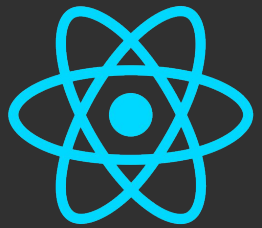
- As **imagens públicas** do nosso projeto podem ficar na pasta public;
- De lá elas podem ser chamadas pelas tags img diretamente pelo **/nome.jpg**;
- Pois **a pasta public fica linkada com o src** das imagens;
- Vamos ver na prática!

ATENÇÃO PARA A NOTA:



Imagens em asset

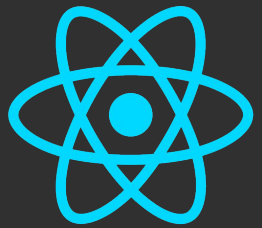
- A pasta public pode ser utilizada para colocar imagens, como fizemos na aula passada;
- Mas um padrão bem utilizada para as imagens dos projetos **é colocar em uma pasta chamada assets**, em src;
- Ou seja, você vai encontrar projetos com as **duas abordagens**;
- Em assets **precisamos importar as imagens**, e **o src é dinâmico** com o nome de importação;



ATENÇÃO PARA A NOTA:

O que são hooks?

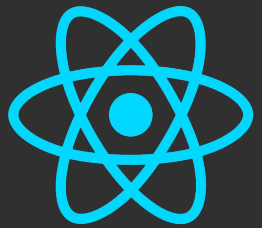
- Recursos do React que tem **diversas funções**;
- Como: **guardar e alterar o estado de algum dado** na nossa aplicação;
- Todos os hooks começam com **use**, por exemplo: **useState**;
- Podemos criar os nossos hooks, isso é chamado de **custom hook**;
- Os hooks precisam ser **importados**;
- Geralmente são úteis em todas as aplicações, **utilizaremos diversos ao longo do curso**;



useState hook

- O hook de **useState** é um dos mais utilizados;
- Utilizamos para **gerenciar o estado de algum dado**, variáveis não funcionam corretamente, o componente não re-renderiza;
- Para guardar o dado definimos o nome da variável e para alterar vamos utilizar **setNome**, onde nome é o nome do nosso dado;
- Vamos ver na prática!

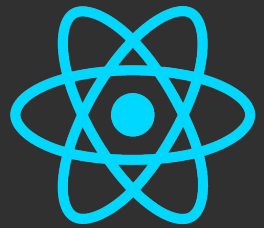
ATENÇÃO PARA A NOTA:



Renderização de lista

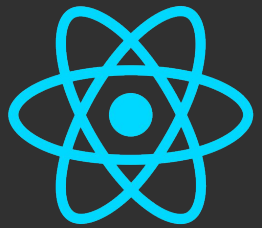
- Uma outra ação bem comum é **renderizar listas** de dados no template;
- Fazemos isso com os dados com tipo de **array**;
- Utilizando o **método map** para nos auxiliar;
- Além dos dados podemos **inserir JSX** em cada iteração;
- Vamos ver na prática!

ATENÇÃO PARA A NOTA:



A propriedade key

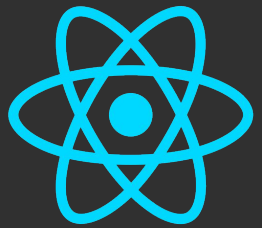
- Iterar listas sem a **propriedade key** nos gera um warning, podemos verificar isso no console;
- **O React precisa de uma chave única** em cada um dos itens iterados;
- Isso serve para **ajudá-lo na renderização do componente**;
- Geralmente teremos um **array de objetos** e podemos colocar key como alguma chave única, como o **id** de algum dado;
- Em **último caso** devemos utilizar o index do método map;



ATENÇÃO PARA A NOTA:

Previous state

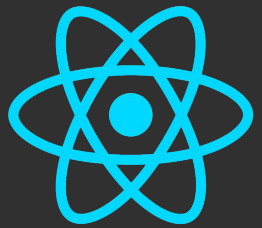
- **Previous state** é um recurso que nos permite pegar o dado em seu valor original dentro de um set de dado;
- **Isso é muito utilizado para modificar listas**, pois temos o valor antigo e transformamos em um valor novo;
- O **primeiro argumento** de um set sempre será o previous state;
- Vamos ver na prática!



ATENÇÃO PARA A NOTA:

Renderização condicional

- **Renderização condicional** é quando imprimimos uma parte do template baseado em uma condição;
- Ou seja, utilizando uma **checagem com if**;
- Isso é interessante em situações como: usuário autenticado/não autenticado;
- Vamos ver na prática!

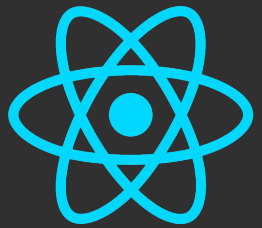


ATENÇÃO PARA A NOTA:

Adicionando um else

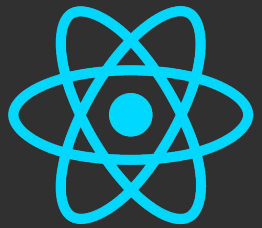
- Podemos também realizar um **if/else no JSX**;
- Aqui devemos utilizar o **if ternário**;
- Onde temos a sintaxe: condição ? bloco1 : bloco2
- Vamos ver na prática!

ATENÇÃO PARA A NOTA:



Props

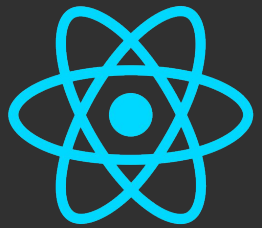
- **Props** é outro recurso fundamental do React;
- Nos permite **passar valores de um componente pai para um componente filho**;
- Isso será muito útil quando os dados forem carregados via banco de dados, por exemplo;
- As props vem em um objeto no **argumento da função do componente**;
- Vamos ver na prática!



ATENÇÃO PARA A NOTA:

Desestruturando props

- É super comum passar **mais de uma prop em um componente**;
- Para facilitar isso o React nos permite **desestruturar as propriedades que chegam**, com o recurso de destructuring;
- Se temos duas props: nage e age;
- Podemos fazer assim function MyComponent(**{name, age}**)
- Agora **não precisamos mais utilizar** props.algumaCoisa;
- Vamos ver na prática!

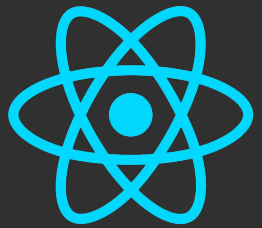


ATENÇÃO PARA A NOTA:

Reutilização de componentes

- Com **props** a **reutilização de componentes** começa a fazer muito sentido;
- Se temos os dados de 1000 carros por exemplo, podemos **reaproveitar o nosso CarDetails 1000 vezes**;
- Isso torna nosso código mais padronizado, facilitando a manutenção;
- Vamos ver na prática!

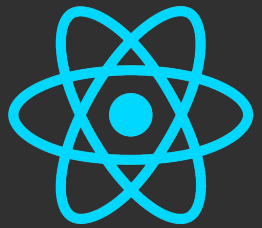
ATENÇÃO PARA A NOTA:



Reutilização com loop

- Os arrays de dados podem ter **muitos itens** também;
- Então o correto é utilizar uma **estrutura de loop (map)** para a sua exibição;
- E com isso conseguimos conciliar os **três conceitos**: renderização de listas, reaproveitamento de componentes e props;
- Vamos ver na prática!

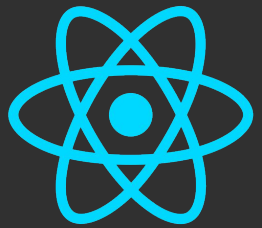
ATENÇÃO PARA A NOTA:



React Fragments

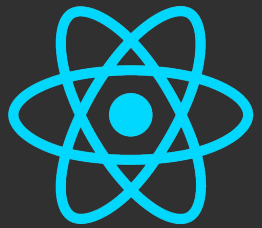
- Os **React fragments** são interessantes para quando precisamos ter mais de um elemento pai em um componente;
- Criamos uma tag vazia: **<>** … **</>**
- **E ela serve como elemento pai**, não alterando a estrutura do HTML com uma div, por exemplo;
- Vamos ver na prática!

ATENÇÃO PARA A NOTA:



Children prop

- **Children prop** é um recurso utilizado para quando um componente precisa ter JSX dentro dele;
- Porém **este JSX vem do componente pai**;
- Então o componente age como um **container**, abraçando estes elementos;
- E children é considerada uma **prop do componente**;
- Vamos ver na prática!

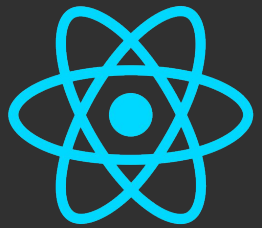


ATENÇÃO PARA A NOTA:

Funções em props

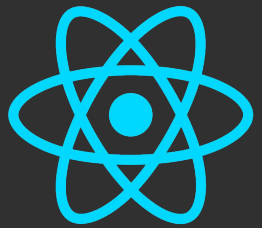
- As **funções podem ser passadas para as props** normalmente;
- Basta criar a função no componente pai e **enviar como prop** para o componente;
- No componente filho ela pode ser ativada por um evento, por exemplo;
- Vamos ver na prática!

ATENÇÃO PARA A NOTA:



Elevação de state

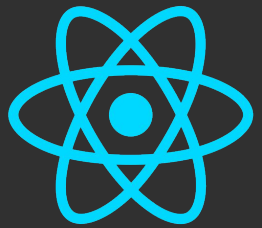
- Elevação de state ou **State lift** é quando um valor é elevado do componente filho para o componente pai;
- Geralmente temos **um componente que usa o state e outro que o altera**;
- Então precisamos passar a alteração para o componente pai, e este passa para o componente que usa o state;
- Vamos ver na prática!

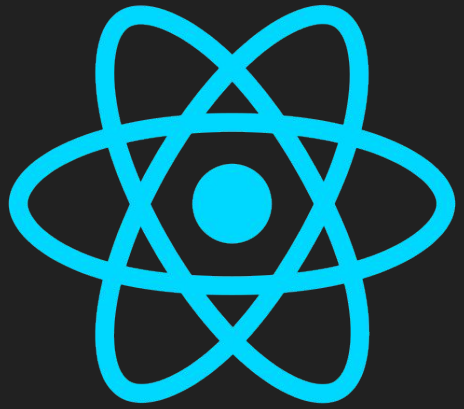


ATENÇÃO PARA A NOTA:

Desafio 4

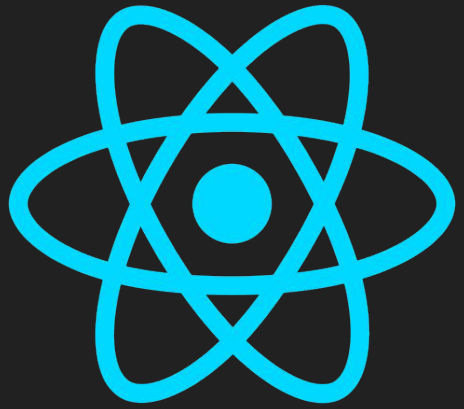
1. Crie um array de objetos compostos de pessoas, com as propriedades de: nome, idade e profissão (array com pelo menos 3 itens);
2. Os dados devem ser exibidos em um componente UserDetails, que você deve criar, todas as informações devem ser exibidas;
3. Faça uma renderização condicional que exibe se o usuário pode tirar carteira de habilitação ou não, imprima isso também no componente;
4. A informação pode ser exibida num parágrafo (checar se idade >= 18);





Avançando no React

Conclusão

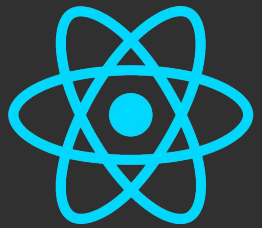


CSS no React

Introdução da seção

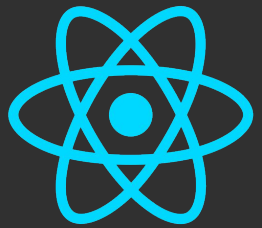
CSS global

- O **CSS global** é utilizado para estilizar diversos elementos em comum ou fazer um reset no CSS;
- Utilizamos o arquivo **index.css** para isso;
- Ele está na pasta src;
- Vamos ver na prática!



CSS de Componente

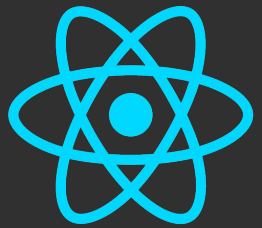
- O **CSS de componente** é utilizado para um componente em específico;
- Geralmente **é criado um arquivo com o mesmo nome do componente** e este é **importado no componente**;
- Note que este método **não é scoped**, ou seja, o CSS vaza para outros componentes se houver uma regra em colisão;
- O React já cria um exemplo desta técnica com o App.css/js;
- Vamos ver na prática!



ATENÇÃO PARA A NOTA:

Inline style

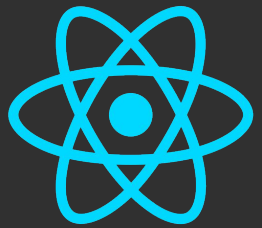
- O inline style do React é **igual o do CSS**;
- Por meio do **atributo style** conseguimos aplicar regras diretamente em um elemento;
- **Devemos optar por outras maneiras de CSS**, o inline pode dificultar a manutenção ou deixar o código imprevisível em algumas situações;
- Vamos ver na prática!



Inline style Dinâmico

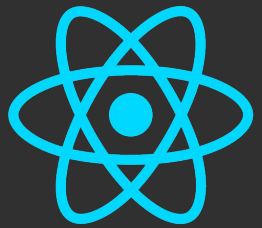
- O **CSS dinâmico inline** aplica estilo baseado em uma condicional;
- Vamos inserir no atributo um **if ternário**;
- Dependendo da condição podemos mudar que regras de estilo um elemento recebe;
- Vamos ver na prática!

ATENÇÃO PARA A NOTA:



Classes dinâmicas no CSS

- Podemos também aplicar lógica para **mudar a classe de CSS de um elemento**;
- Também utilizaremos o **if ternário**;
- Essa abordagem é **mais interessante que o CSS inline**;
- Pois as classes estarão isoladas no arquivo de CSS, resolvendo o problema de organização de código;
- Vamos ver na prática!

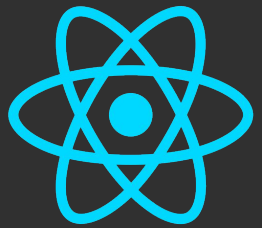


ATENÇÃO PARA A NOTA:

CSS Modules

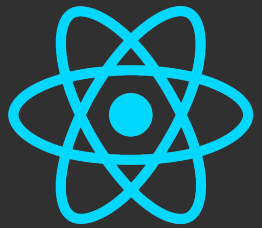
- O **CSS Modules** é um recurso de CSS **scoped**;
- Ou seja, ele vai ser **exclusivo do componente**;
- O nome do arquivo é: **Componente.module.css**;
- Precisamos importá-lo também no componente;
- Vamos ver na prática!

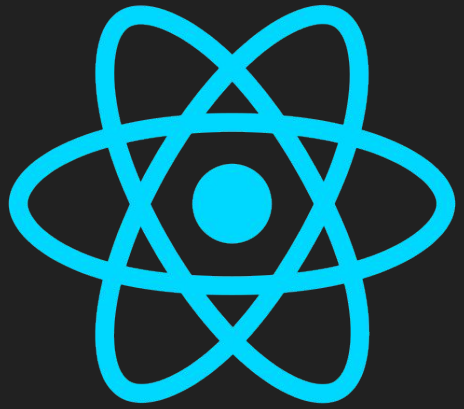
ATENÇÃO PARA A NOTA:



Desafio 5

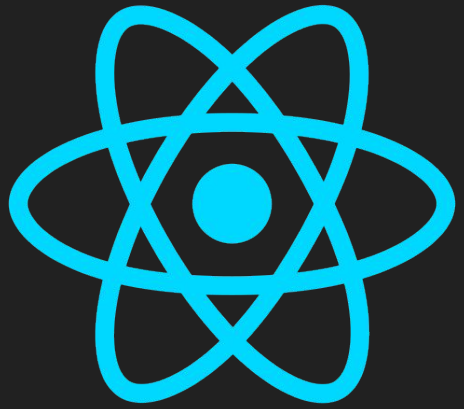
1. Crie um novo projeto chamado challengecss;
2. No CSS global zere a margin, padding e coloque uma fonte que você goste;
3. Crie um componente que exibe detalhes de carros, este componente deve ser estilizado com scoped;
4. Exiba pelo menos 3 carros;
5. Coloque um título em App.js para o seu projeto, estilize com o App.css;





CSS no React

Conclusão



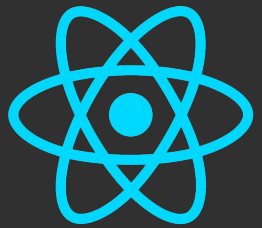
React e formulários

Introdução da seção

Formulários e React

- No React vamos também utilizar a **tag form** para formulários;
- As labels dos inputs contém o atributo **htmlFor**, que deve ter o valor do name do input;
- **Não utilizamos action**, pois o processamento será feito de form assíncrona;
- Vamos criar um form!

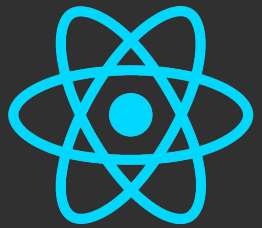
ATENÇÃO PARA A NOTA:



Label envolvendo o input

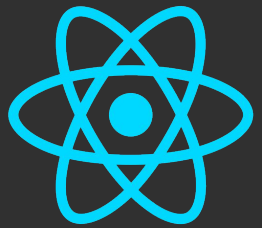
- Em React um padrão comum é a **tag label envolvendo o input**;
- Isso faz com que o atributo for se torne **opcional**;
- **Simplificando nossa estrutura de HTML**, sem perder a semântica;
- Vamos ver isto na prática!

ATENÇÃO PARA A NOTA:



Manipulação de valores

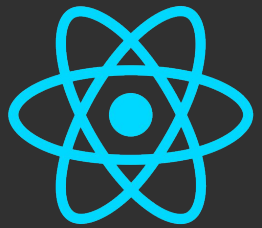
- Para manipular os valores dos inputs vamos utilizar o **hook useState**;
- Ou seja, podemos armazenar na variável **e utilizar o set para alterar o valor**;
- Vamos criar uma função para alterar o valor no evento **onChange**;
- Deixando nosso código fácil de trabalhar nas próximas etapas: como envio dos dados para BD e validação;
- Vamos ver isto na prática!



ATENÇÃO PARA A NOTA:

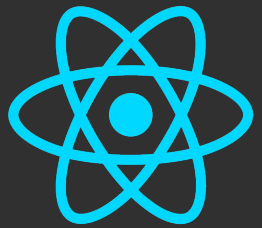
Simplificando a manipulação

- Quando temos vários inputs podemos **realizar a manipulação de forma mais simples**;
- Basicamente criamos uma **função inline no onChange**;
- Ela vai **alterar o valor do state** com o método set, da mesma forma que a função isolada;
- Vamos ver isto na prática!



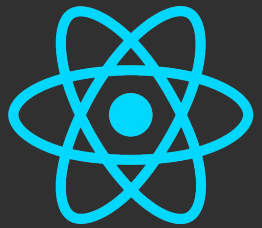
Envio de formulário

- Para enviar um form vamos utilizar o evento **onSubmit**;
- **Ele chamará uma função**, e nesta devemos lembrar de parar a submissão com o **preventDefault**;
- Nesta etapa podemos realizar validações, envio de form para o servidor, reset de form e outras ações;
- Vamos ver isto na prática!



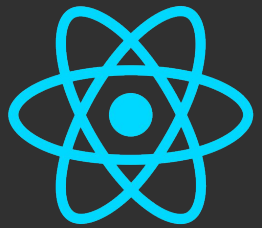
Controlled inputs

- **Controlled inputs** é um recurso que nos permite mais flexibilidade nos forms de React;
- Precisamos apenas **igualar o valor ao state**;
- Um uso muito comum: formulários de edição, que os dados vem do back-end, conseguiremos preencher o input mais facilmente;
- Vamos ver isto na prática!



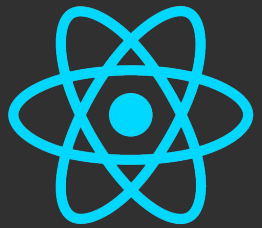
Limpando formulários

- Com o controller inputs limpar o form será **fácil**;
- Basta **atribuir um valor de uma string vazia aos states** e pronto!
- Isso será feito após o envio, em formulários que o usuário precisa preencher novamente;
- Vamos ver isto na prática!



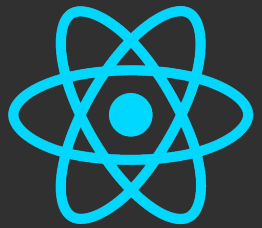
Input de Textarea

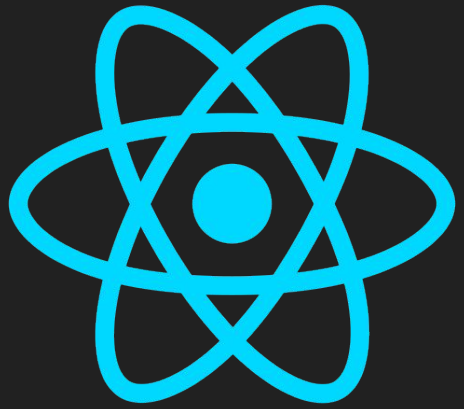
- O textarea **pode ser considerado um input de texto** normal;
- Utilizaremos o **value** para alterar o state inicial;
- E o evento **onChange** para modificar o valor do state;
- Vamos ver isto na prática!



Input de Select

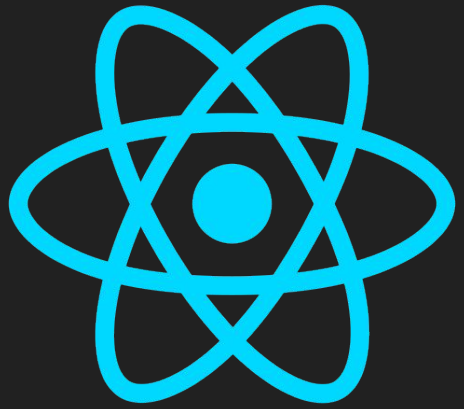
- O select também será **muito semelhante** aos outros inputs;
- Quando temos a alteração de um valor o **evento onChange** pode captar isso;
- O value também pode atribuir qual **option** estará selecionada;
- Vamos ver isto na prática!





React e formulários

Conclusão

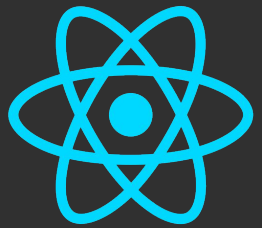


Requisições HTTP e React

Introdução da seção

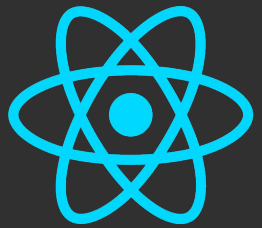
JSON server

- O **JSON server** é um pacote npm;
- Ele **simula uma API**, e isso nos possibilita fazer requisições HTTP;
- Vamos aprender a **integrar este recurso com o React**;
- Podemos entender isso como uma etapa de preparação para APIs reais;
- Ou seja, atingir o mesmo resultado mas sem precisar de uma estrutura no back-end;
- Vamos criar um projeto e instalar o JSON server;



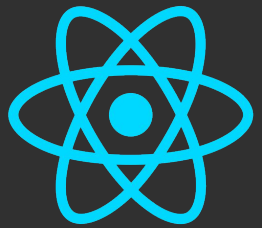
A importância do useEffect

- O **useEffect** faz com que determinada ação seja executada apenas uma vez;
- Isso é interessante pois os componentes estão sempre se **re-renderizando**, então precisamos ter **ações únicas** às vezes;
- O useEffect ainda possui um **array de dependências**, que deve conter os dados que ativem a execução da função de forma automática;
- O useEffect estará presente sempre nas **requisições assíncronas**!



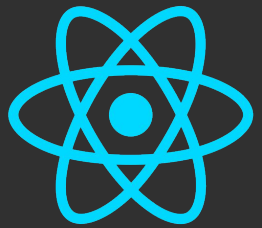
Resgatando dados com React

- Para trazer os dados vamos ter que utilizar vários recursos;
- Primeiramente ter um local para salvá-los (**useState**);
- Renderizar a chamada a API apenas uma vez (**useEffect**);
- Um meio de fazer a requisição assíncrona (**Fetch API**);
- Vamos ver isto na prática!



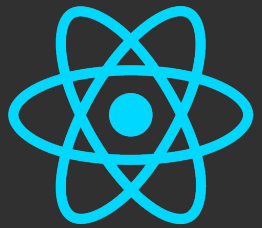
Adicionando dados

- Para adicionar um item vamos precisar resgatar os dados do form com o **useState**;
- Reunir eles em uma **função após o submit** e enviar um request de **POST** para a nossa API;
- O processo é bem parecido com o de resgate de dados, mas agora estamos **enviando dados**;
- Vamos ver isto na prática!



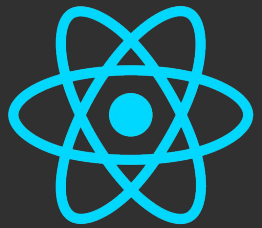
Carregamento dinâmico de dados

- Se a requisição foi feita com sucesso, podemos **adicionar o item a lista após o request**;
- Isso torna nossa aplicação mais **performática**;
- Utilizaremos o **set do useState** para isso;
- Vamos ver isto na prática!



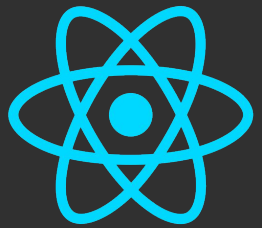
Custom hook para o fetch

- É normal dividir funções que podem ser reaproveitadas em hooks;
- Esta técnica é chamada de **custom hook**, e vamos criar um para o resgate de dados;
- Os hooks geralmente ficam na **pasta hooks**;
- Devemos utilizar o padrão **use**Name;
- Basicamente criamos uma função e exportamos ela;
- Vamos ver isto na prática!



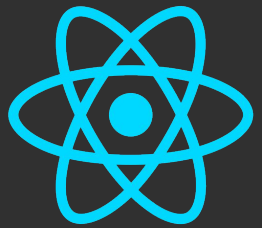
Refatorando o POST

- Podemos **utilizar o mesmo hook** para incluir uma etapa de POST;
- **Vamos criar um novo useEffect** que mapeia uma outra mudança de estado;
- Após ela ocorrer executamos a adição de produto;
- **Obs:** nem sempre reutilizar um hook é a melhor estratégia;
- Vamos ver isto na prática!



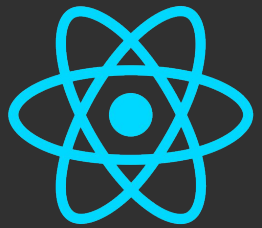
Estado de loading

- Quando fizermos requisições para APIs **é normal que haja um intervalo de loading entre a requisição e o recebimento** da resposta;
- Podemos fazer isso no nosso **hook** também;
- **Identificar quando começa e termina** este estado;
- Vamos ver isto na prática!



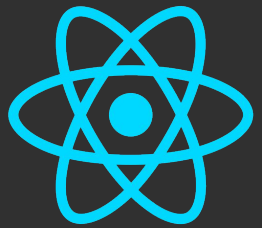
Estado de loading no POST

- Podemos bloquear ações indevidas em outras requests também, **como no POST**;
- Uma ação interessante é **remover a ação de adicionar outro item** enquanto o request ainda não finalizou;
- Vamos ver isto na prática!



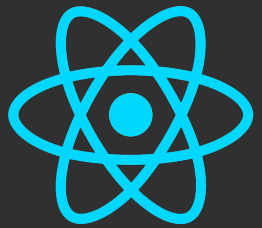
Tratando erros

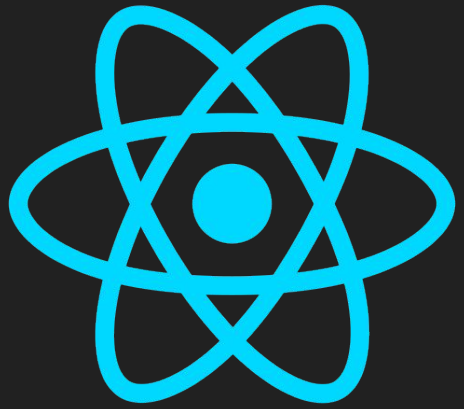
- Podemos tratar os erros das requisições por meio de um **try catch**;
- Além de pegar os dados do erro, também podemos **alterar um state para imprimir um elemento se algo der errado**;
- Desta maneira conseguimos **prever vários cenários** (dados resgatados, carregamento e erro);
- Vamos ver isto na prática!



Desafio 6

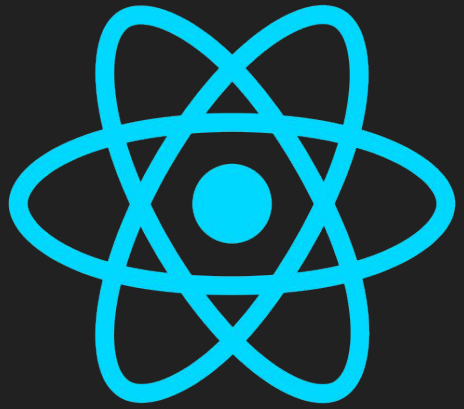
1. Crie um botão nos produtos;
2. Este botão deve disparar uma função de remoção de produto;
3. A URL deve ser a mesma da API + o id do produto: products/1
4. Você vai precisar identificar quando é uma requisição de DELETE, para mudar o verbo a http das configurações;
5. Utilize a ideia do método de POST para derivar para o DELETE, pode ser com if/else;





Requisições HTTP e React

Conclusão

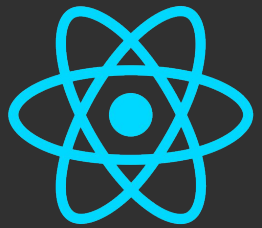


React Router

Introdução da seção

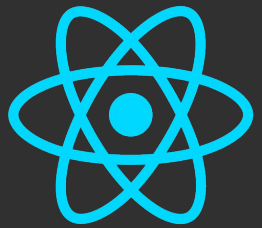
O que é React Router?

- **React Router** é um dos pacotes mais utilizados para criar uma estrutura de rotas em aplicações de React;
- Ou seja, permite que nossas **SPAs tenham múltiplas páginas**;
- Precisamos **instalar** no nosso projeto;
- A **configuração e utilização** é simples;
- Também temos outras funções como: **Redirect**, **Nested Routes**, **Not Found Routes** e outros;



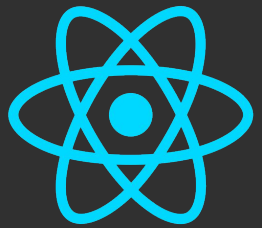
Desafio 7

1. Crie um novo projeto para esta unidade;
2. Instale o módulo do React Route neste projeto, que é o: react-router-dom
3. Instale também o json-server
4. Coloque pelo menos três produtos da seção anterior no seu arquivo de db.json
5. Crie um script para inicializar o json-server



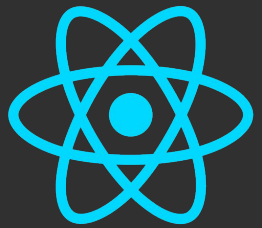
Configurando o React Router

- Para configurar o React Router vamos ter que importar três elementos de **react-router-dom**;
- **BrowserRouter**: Define onde a área do nosso app que vai trocar as páginas;
- **Routes**: Define as rotas;
- **Route**: um elemento deste para cada rota, configurar com path e componente da rota;



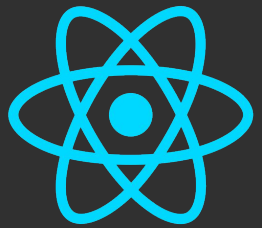
Adicionando links

- Para criar links para as páginas vamos precisar utilizar o **Link** do React Router;
- No Link configuramos o parâmetro **to**, que recebe a **URL/path** que será redirecionado quem clicar no link;
- Vamos criar um componente de Navbar para isso;



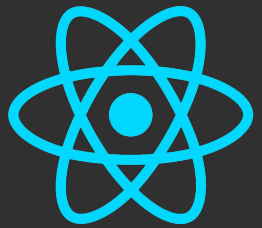
Carregando dados

- Vamos exercitar novamente o carregamento de dados com nosso hook **useFetch**;
- Depois poderemos utilizá-los para o **carregamento de dados individuais**;
- Utilizaremos o hook igual ao da última seção e vamos imprimir os produtos na Home da mesma forma;



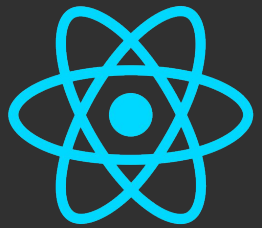
Rota dinâmica

- Para criar uma rota dinâmica vamos precisar definir uma **nova Route** em App.js;
- Que deve ter o padrão de: **/products/:id**
- Onde **:id** é o dado dinâmico, ou seja, podemos ter qualquer valor;
- Na página podemos utilizar o hook **useParams** para resgatar esta informação;
- Vamos ver na prática!



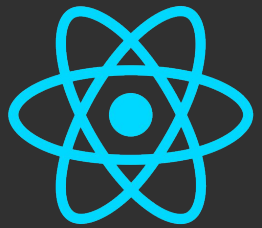
Carregando dado individual

- Graças ao passo dado na aula passada o **carregamento individual** de um produto será fácil;
- Vamos utilizar o id recebido para **formar a nova URL**;
- E por fim podemos utilizar o hook **useFetch** para trazer o item;
- Por fim faremos a **validação e impressão** do mesmo no JSX;



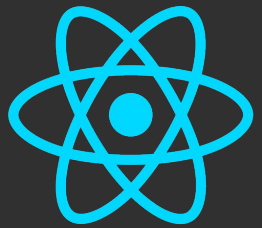
Nested route

- As nested routes indicam **URLs mais complexas**, como: **/products/:id/something**;
- Neste caso vamos precisar criar um componente que corresponda com o padrão indicado e também a URL em App.js;
- Na nested route **teremos o acesso ao parâmetro da URL** também;
- Vamos ver na prática!



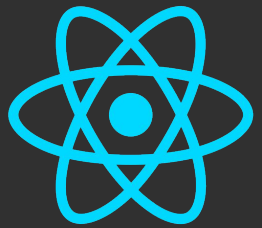
No match route (404)

- Podemos criar uma **página 404** facilmente com o React Router;
- Basta **criarmos o componente** da página;
- E no arquivo App.js definir um **path como ***;
- Desta maneira, qualquer rota que não exista cairá neste componente;
- Vamos ver na prática!



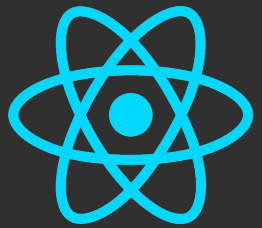
Link ativo

- Para ter fácil acesso a uma modificação para os links ativos vamos trocar o Link pelo **NavLink**;
- Neste elemento temos acesso a um valor chamado **isActive**;
- Ou seja, podemos **ativar uma classe** se a rota atual for a que está no **atributo to**;
- Vamos ver na prática!



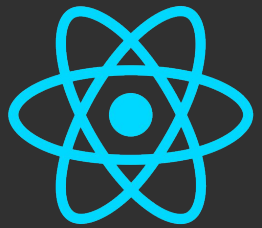
Search Params

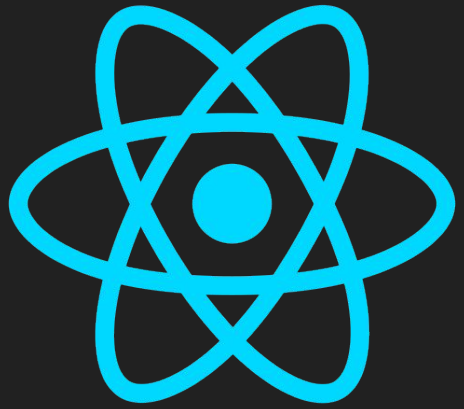
- **Search Params** é um recurso que permite obter o que vem na URL em forma de parâmetro, ex: produtos?**q=camisa**
- Utilizamos o hook **useSeachParams** para obtê-los;
- Com este recurso fica simples fazer uma **funcionalidade de busca** no sistema;
- Vamos ver na prática!



Redirect

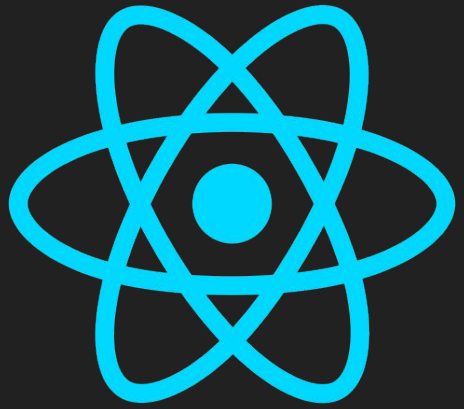
- Podemos precisar de um **redirecionamento de páginas** eventualmente;
- **Exemplo:** uma página antiga do sistema responde agora a uma nova URL;
- Para isso vamos **criar a rota com Route** normalmente;
- Mas em element vamos utilizar o **componente Navigate** com um **to** que vai para a rota correta;
- Vamos ver na prática!





React Router

Conclusão

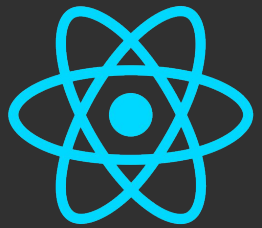


Context API

Introdução da seção

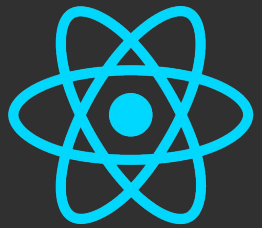
O que é Context API?

- Um recurso que facilita o **compartilhamento de um estado entre componentes**;
- Ou seja, quando precisamos de **dados 'globais'**, provavelmente precisamos utilizar o Context;
- **O Context precisa encapsular os componentes** que receberão seus valores, geralmente colocamos no App.js ou index.js;
- Os contextos geralmente ficam na **pasta context**;



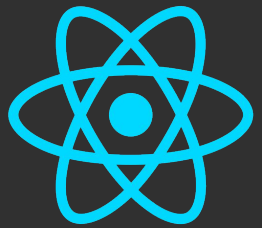
Desafio 8

1. Crie um novo projeto para trabalharmos com context;
2. Este projeto deve ter instalado o react router;
3. Crie 3 páginas;
4. Faça uma navbar e coloque o link para as três;



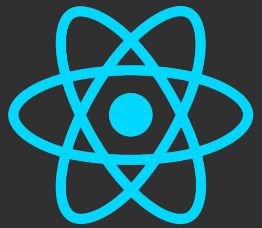
Criando o contexto

- Primeiramente vamos ter que **criar o Context**;
- O arquivo deve sempre ter a **primeira letra maiúscula** no nome, e geralmente termina em Context: SomeContext.js;
- A convenção é deixar na **pasta context** em src;
- Onde vamos utilizar o valor do contexto, **o arquivo precisa ser importado**;



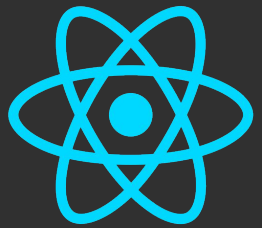
Criando o provider

- O Provider vai **delimitar onde o contexto é utilizado**;
- Vamos criar uma espécie de componente com a **prop children**;
- E este Provider deve **encapsular os demais componentes** em que precisamos consultar ou alterar o valor;
- Geralmente ele fica em **App.js** ou em **index.js**;
- Agora poderemos **compartilhar o valor do contexto** em todos os componentes;



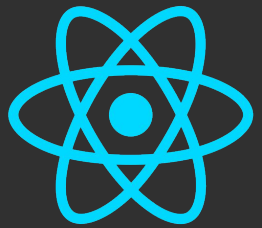
Alterando o contexto

- Para alterar o valor do contexto **precisamos criar um componente que utilize a função da mudança de contexto**;
- Esta mudança ocorrerá no Context e **poderá ser consumida por todos os componentes** que recebem o contexto;
- E assim finalizamos o **ciclo** da Context API;
- Vamos ver na prática!



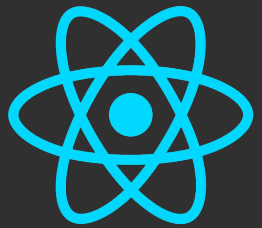
Refatorando context com hook

- Podemos **criar um hook para utilizar o contexto**, isso nos dá algumas vantagens;
- **Não precisamos importar o useContext em todos os lugares** que vamos usar o contexto, só o hook;
- Temos um espaço para fazer uma **validação do contexto**;
- Vamos ver na prática!



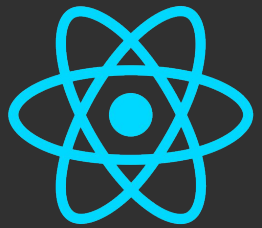
Contexto mais complexo

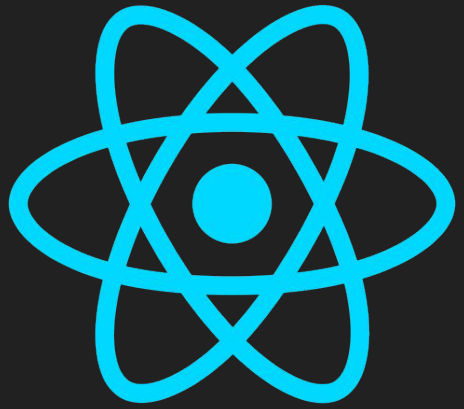
- Contextos mais complexos podem ter **variações no comportamento**;
- Para isso vamos utilizar um hook chamado **useReducer**;
- Que **é como um useState**, mas para controle de dados complexos;
- No reducer teremos diferentes ações com um **switch**;
- E na aplicação vamos consumir o estado atual do dado que está no reducer;
- Vamos ver na prática!



Alterando contexto complexo

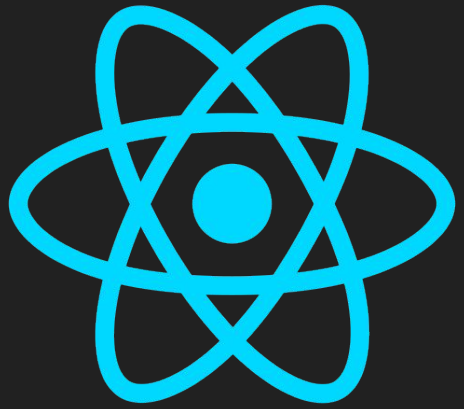
- Para alterar o contexto vamos utilizar uma função chamada **dispatch**;
- Ela estará no **reducer** também;
- E deve conter todas as informações necessárias para a alteração do valor do contexto;
- Ou seja, o **switch entra em ação** e retorna um novo contexto;
- Vamos ver na prática!





Context API

Conclusão da seção

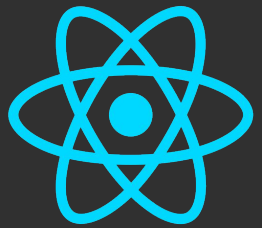


Os hooks do React

Introdução da seção

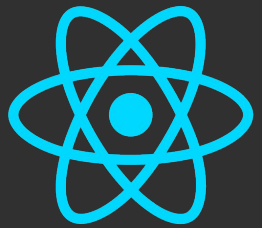
useState

- O **useState** é um dos principais hooks do React;
- O principal propósito é **gerenciar valores**;
- Poderemos **consultar** um valor e **alterar**;
- Isso nos permite **re-renderizar um componente**, o que não acontece com a manipulação de variáveis;
- Vamos ver na prática!



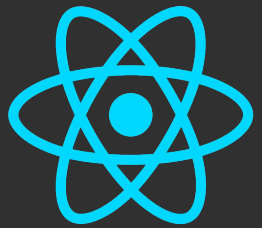
useState e inputs

- **Atrelando o useState a um input** podemos fazer algumas ações;
- Alteração de um state por evento de **onChange**;
- Limpeza de inputs (**Controlled Input**);
- Após preenchimento total do form, unir os states e fazer um envio dos dados para o back-end;
- Vamos ver na prática!



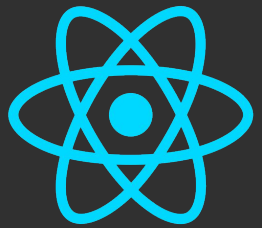
useReducer

- O **useReducer** tem a mesma função que o useState, ele gerencia valores;
- Porém temos a possibilidade de **executar uma função na hora da alteração do valor**;
- Então temos que o useReducer recebe **um valor** para gerenciar e **uma função** para alterar este valor;
- Vamos ver na prática!



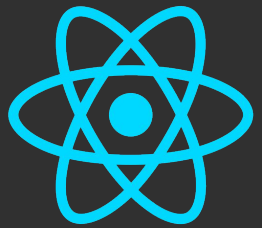
Avançando em useReducer

- Se o useReducer fosse utilizado como no exemplo passado, **não teria tanta diferença do useState**;
- Por isso o reducer geralmente contém **operações mais complexas**, utilizando a estrutura **switch com actions**;
- Esta situação foi apresentada na seção de Context API;
- Vamos ver na prática!



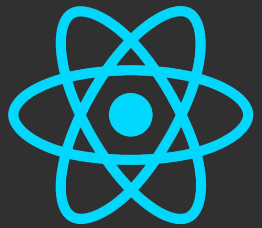
useEffect

- O **useEffect** é utilizado para várias ações no nosso App, junto com useState é um dos hooks mais utilizados;
- Podemos realizar desde **alterações na DOM a requisições HTTP**;
- E o grande motivo é: conseguimos **controlar quantas vezes algo acontece**;
- A sintaxe é formada por **uma função a ser executada** e **um array de dependências**;
- Vamos ver na prática!



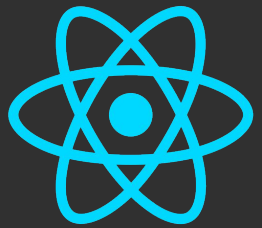
useEffect com array vazio

- Uma estratégia interessante para algumas situações é utilizar o **useEffect apenas uma vez**;
- Para isso apenas precisamos deixar o **array de dependências vazio**;
- **Ao renderizar o componente** a lógica será executada;
- Vamos ver na prática!



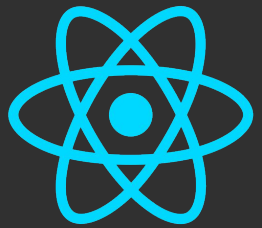
Array de dependências do useEffect

- Outra maneira de controlar quando o useEffect será executado é **colocando algum item no array de dependências**;
- Assim sempre que o item for alterado, teremos o useEffect sendo executado novamente;
- Nos fornecendo um **maior controle de quando a função deve ou não ser executada**;
- Vamos ver na prática!



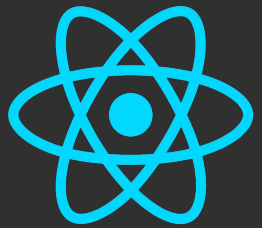
Limpeza do useEffect

- Alguns efeitos precisam ter uma **técnica de cleanup (limpeza)** para garantir o seu funcionamento;
- Não fazer isso pode gerar **erros ou comportamentos indesejados**;
- **Exemplo:** um timeout que ao mudar de página pode continuar a ser executado, pela falta desta limpeza;
- Vamos ver na prática!



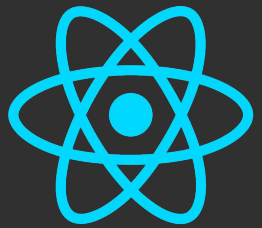
useContext

- O **useContext** é o hook utilizado para consumir um contexto, da Context API;
- Vamos precisar **criar o contexto** e também o **Provider**;
- **Envolver os componentes** que receberão os valores compartilhados;
- E então fazer o uso do hook onde necessário;
- Vamos ver na prática!



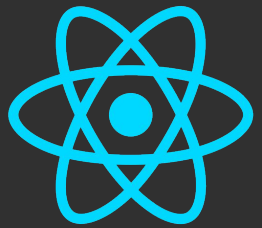
useRef

- O **useRef** pode ser utilizado como useState para gerenciar valores;
- A diferença é que ele é um **objeto**, seu valor está na propriedade **current**;
- Outra particularidade do useRef é que ele **não re-renderiza o componente ao ser alterado**, sendo interessante alguns casos;
- Vamos ver na prática!



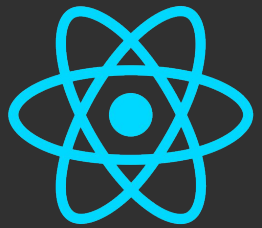
useRef e o DOM

- O useRef pode ser utilizado para **selecionar elementos do JSX**;
- Com isso podemos fazer **manipulação de DOM ou aplicar funções como a focus**, que foca no input;
- Este é um outro uso muito interessante para este hook;
- Vamos ver na prática!



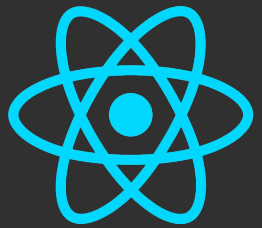
useCallback

- O hook de **useCallback** pode ser utilizado para duas situações;
- Ele basicamente **memoriza uma função**, fazendo ela **NÃO ser reconstruída a cada renderização** de um componente;
- O primeiro caso é quando estamos prezando pela **performance**, então queremos que uma função muito complexa não seja reconstruída toda vez;
- Já o segundo é quando **o próprio React nos indica que uma função deveria estar contida em um useCallback**, para evitar problemas de performance;



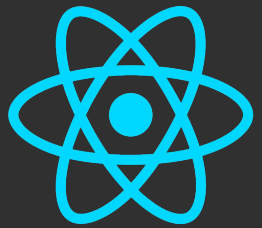
useMemo

- O **useMemo** pode ser utilizado para garantir a **referência de um objeto**;
- Fazendo com que algo possa ser atrelado a uma referência e não a um valor;
- Com isso conseguimos **condicionar useEffects a uma variável** de maneira mais inteligente;
- Vamos ver na prática!



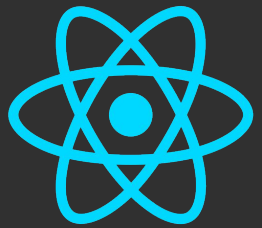
useLayoutEffect

- Muito parecido com o **useEffect**;
- A grande diferença é que este hook **roda antes de renderizar o componente**;
- Ou seja, o hook é **síncrono**, bloqueando o carregamento da página para o sucesso da sua funcionalidade;
- A ideia é executar algo antes que o usuário veja a página;
- Vamos ver na prática!



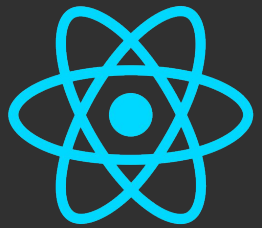
useImperativeHandle

- Com o hook **useImperativeHandle** temos como acionar ações em um outro componente de forma imperativa;
- Como não podemos passar refs como props, precisamos usar uma função **fowardRef**;
- Isso nos permite passar as referências, e torna o nosso exemplo viável;
- Vamos ver na prática!



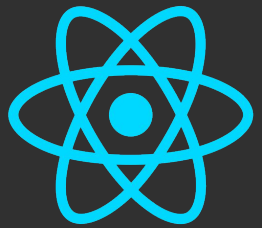
Custom hooks

- Os **custom hooks** são os hooks que nós criamos;
- Muitas vezes para **abstrair funções complexas do componente** ou simplesmente **reaproveitar código**;
- Esta técnica é muito utilizada em projetos profissionais;
- Vamos ver na prática!



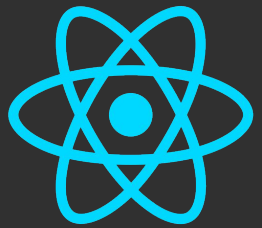
React Dev Tools

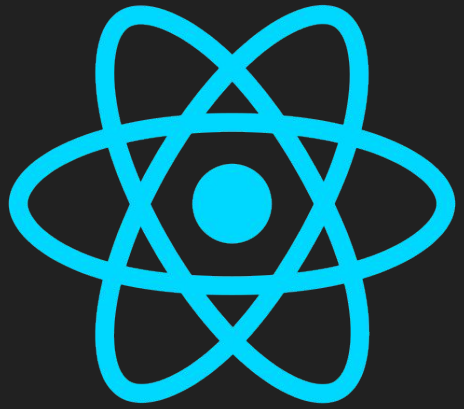
- **React Dev Tools** é uma extensão para o navegador;
- Nela conseguimos **entender o que o React está gerando** por meio do nosso código;
- Conseguimos também **verificar os states e outros parâmetros**;
- Vamos instalar!



useDebugValue

- É um hook que é utilizado para **debug**;
- Aconselhado para ser **utilizado em custom hooks**;
- Adiciona uma área no **React Dev Tools**, ela estará no componente em que o hook é utilizado;
- Vamos ver na prática!





Os hooks do React

Conclusão da seção