**BIBLIOTECAS**

**Sumário**

[**1. React 2**](#_heading=h.3znysh7)

[Instalação 2](#_heading=h.2et92p0)

[Vite 2](#_heading=h.tyjcwt)

[Create react app 3](#_heading=h.3dy6vkm)

[Práticas de projetos 4](#_heading=h.1t3h5sf)

[1. Descrevendo UI 4](#_heading=h.4d34og8)

[1.0 Primeiro Componente 4](#_heading=h.2s8eyo1)

[1.1 Importando e exportando componentes 5](#_heading=h.17dp8vu)

[1.2 Escrevendo marcação com JSX 7](#_heading=h.3rdcrjn)

[1.3 JavaScript em JSX com chaves 8](#_heading=h.26in1rg)

[1.4 Passando props para um componente 11](#_heading=h.lnxbz9)

[1.5 Renderização condicional 16](#_heading=h.35nkun2)

[1.6 Lista de renderização 19](#_heading=h.1ksv4uv)

[1.7 Mantendo os componentes puros 23](#_heading=h.44sinio)

[2. Adicionando interatividade 24](#_heading=h.2jxsxqh)

[2.0 Respondendo a eventos 27](#_heading=h.z337ya)

[2.1 Estado: Memória de um componente 32](#_heading=h.3j2qqm3)

[2.2 CONTINUAR DAQUI Renderizar e confirmar 36](#_heading=h.1y810tw)

[2.3 Estado com um instantâneo 37](#_heading=h.4i7ojhp)

[2.4 Enfileirando uma série de atualizações de estado 38](#_heading=h.2xcytpi)

[2.5 Atualizando objetos no estado 40](#_heading=h.1ci93xb)

[2.6 Atualizando matrizes no estado 45](#_heading=h.3whwml4)

[3. Gerenciando estados 53](#_heading=h.2bn6wsx)

[3.0 Reagindo à entrada do estado 63](#_heading=h.qsh70q)

[Etapa 2: determinar o que desencadeia… https://react.dev/learn/reacting-to-input-with-state 65](#_heading=h.3as4poj)

[Framework 65](#_heading=h.1pxezwc)

[Next.js 65](#_heading=h.49x2ik5)

[Remix 65](#_heading=h.2p2csry)

[Gatsby 66](#_heading=h.147n2zr)

[Expo 66](#_heading=h.3o7alnk)

[**1. Axios Introdução 66**](#_heading=h.ihv636)

[**2. Instalação 67**](#_heading=h.32hioqz)

[**3. Exemplos Básicos 67**](#_heading=h.1hmsyys)

[**4. Solicitações POST 67**](#_heading=h.41mghml)

[4.0 Formulários 68](#_heading=h.2grqrue)

[**5. API Axios 69**](#_heading=h.vx1227)

[**6. Instâncias Axios 70**](#_heading=h.3fwokq0)

[**7. Configuração de requisição 70**](#_heading=h.1v1yuxt)

[**8. Esquema de Resposta 74**](#_heading=h.4f1mdlm)

[**9. Configurações de Padrões 74**](#_heading=h.2u6wntf)

[**10. Interceptadores 75**](#_heading=h.19c6y18)

[**11. Manipuladores de Erros 76**](#_heading=h.3tbugp1)

[**12. Cancelamento 77**](#_heading=h.28h4qwu)

[12.0 AbortController 77](#_heading=h.nmf14n)

[12.1 CancelToken (DESCONTINUADO) 78](#_heading=h.37m2jsg)

[**13. Corpo de Codificação de URL 78**](#_heading=h.1mrcu09)

# **1. React**

Biblioteca JavaScript que serve para renderizar interfaces de usuário (UI).

**Aplicativos feitos de componentes** (parte de interface do usuário - função javascript) que possui sua própria lógica e aparência.

* funções (**nomes devem começar com letra maiúscula, react reconhece letra minúscula como uma tag html**) que retornam marcações .
* componentes **reutilizáveis e encaixáveis**.
* construído a partir de partes isoladas.
* permite combinação de CSS e javascript em componentes reutilizáveis.

**Nota: instalar Node para desenvolvimento local.**

**OBS: suporte para exportação htlm/css/js estática**.

**Frameworks** - next.js, remix, gatsby e expo.

* dimensionamento de acordo com necessidades.

## **Instalação**

### **Vite**

Configure um ambiente JS modular que permita escrita de react em arquivos individuais. Utilize **Vite** para configuração:

* **node 14.18 ou superior**.
* desenvolvimento mais rápido e enxuto para projetos web.
* **Composição** - aprimoramento de recursos avançados em módulos nativos e compilação agrupada do código com Rollup (pré-configurado)
* Apresente api **plug-in** e **api javascript**.
* Montagem de projeto - digite no terminal **npm create vite@latest**.
  + formato - **npm create vite@latest nomeProjeto -- --template modelo**.
  + **modelos aceitos** - vanilla, vanilla-ts, vue, vue-ts, react, react-ts, react-swc, react-swc-ts, preact, preact-ts, lit, lit-ts, svelte, svelte-ts.
* **Vite** servidor e **index.html** é ponto de entrada.
* **<root>** - diretório raiz onde arquivos são servidos.
* digite **npx vite**.

### **Create react app**

Iniciar projeto com node digitando **npm init**. Depois, instale o criador de aplicativos react com **npm i create-react-app nomeProjeto**.

**OBS: CASO JÁ TENHA INSTALADO, DESINSTALE COM npm uninstall -g create-react-app**.

Ao finalizar o projeto digite **npm run build** para implantar na produção.

* otimiza compilação para melhor desempenho.

Utilize **npm test** para verificar alguns testes de modo interativo.

**suporte para:** React, JSX, ES6, TypeScript e Flow.

ferramentas são pré-configuradas para funcionar de uma maneira específica.

**serve para aplicativos react de página única e aprender a usar a biblioteca.**

Para renderizar servidor com react e node é **importante a utilização next.js**. Pois, é independente do back-end.

Use **nwb** para **integrar o código React com uma estrutura de modelo do lado do servidor** ou s**e não estiver construindo um aplicativo de página única.**

## **Práticas de projetos**

Quiz - <https://www.youtube.com/watch?v=HlkbeikH8cs>. CONCLUÍDO.

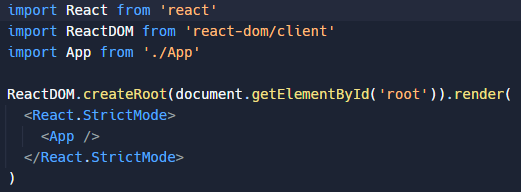
## **1. Descrevendo UI**

### **1.0 Primeiro Componente**

Comunidades de componentes reutilizáveis: **Chakra UI** e **Material UI**.

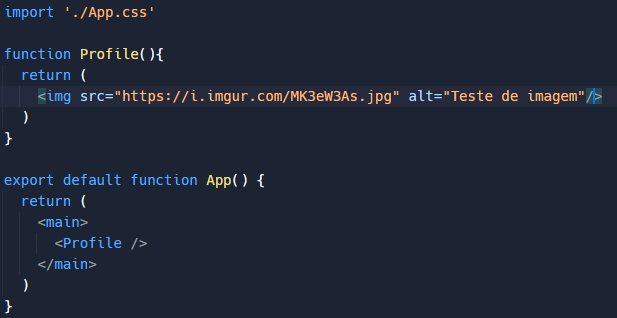
**Ex: index.jsx - principal**

**obs: extensão jsx foi criado para formar elementos para serem utilizáveis com templates de aplicações React.**



**Faz a renderização dos componentes**.

**App.jsx - componentes**



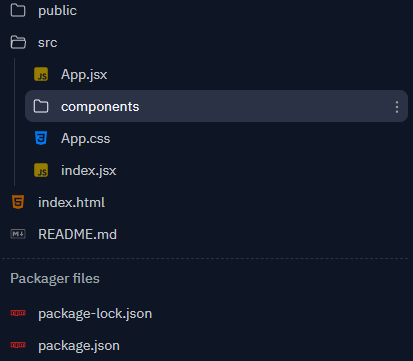
**Componentes criados com a utilização de funções**.

**Componentes só podem renderizar outros componentes**, mas não as suas definições.

**Quando criado um projeto react um componente raiz será criado automáticamente**. Por exemplo, no create react app o componente será definido em **src/app.js** e no next.js será definido em **pages/index.js**.

### **1.1 Importando e exportando componentes**

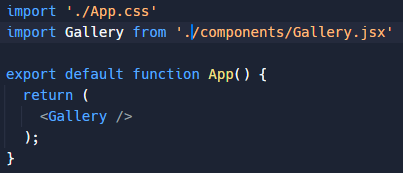
**organização de arquivos**:



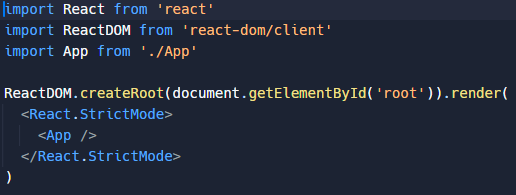
**formato de importação**:



**arquivo gallery dentro de componentes**.

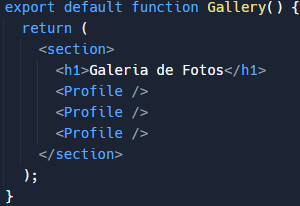


**arquivo react raiz**.



**Tipos de exportação**:

* exportação padrão - **export default function nome(){ … }**



**exportando Gallery de forma padrão**.



**importando Gallery. O Nome após import pode ser qualquer um**.

* exportação nomeada - **export function nome(){ … }**



**Exportando Gallery de forma nomeada**.



**Importando Gallery a partir de seu nome padrão**.

### **1.2 Escrevendo marcação com JSX**

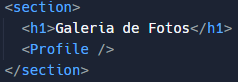
extensão de sintaxe para JS que **permite marcações semelhantes ao HTML** dentro do JS.

* informações dinâmicas

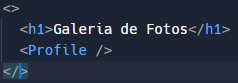
Evolução da lógica JS para determinar o conteúdo HTML - **lógica de rendenrização e marcação**.

**Regras JSX**:

* envolva elementos de um componente em um único pai.



**section está agindo como pai dos elementos h1 e do componente profile**.



**Esta é outra forma de envolver elementos sem que haja adição de marcação**.

**Obs: a tag vazia é um fragmento que agrupa coisas sem deixar rastros na árvore HTML**.

* **tags explicitamente fechadas**: tag de fechamento automático **<img/>** e tag de empacotamento **<li> … </li>**.
* **CamelCase**: atributos escritos em JSX se tornam chaves de objetos JS. Portanto, atributos HTML e SVG tão em camelCase.

Ex: **stroke-width** → **strokeWidth**.

### **1.3 JavaScript em JSX com chaves**

Strings ao serem passadas como atributos para o JSX, devem ser colocadas entre aspas simples ou duplas.

**exemplo**:

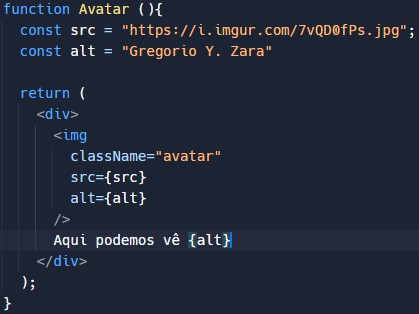


**outra forma (dinâmica)**,



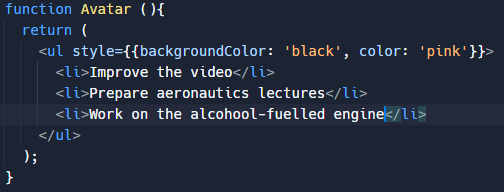
**chaves permitem trabalhar com js mesmo com sua marcação**.

**Incorporação de chaves dentro de escrita HTML:**

****

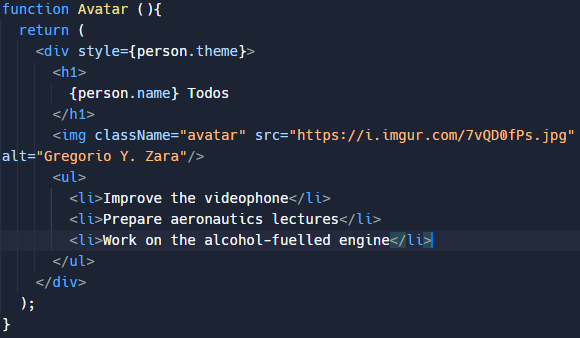
**Modos de usar**:

* dentro de elementos (tags).
* usados como atributos.
* **OBS: não pode ser usado como tag ou dentro de aspas**.
* **se passar algum objeto** é importante que o mesmo seja envolvido com outro par de chaves.
  + Exemplo:



**OBS: propriedades inline style são escritas em camelCase. ex: Em html <ul style=”background-color: black”> e em JSX <ul style={{ backgroundColor:’black’ }}>.**

**Exemplo: Movendo várias expressões de uma vez**.

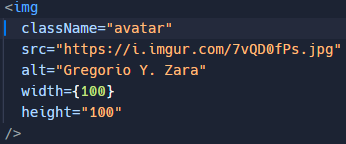


### **1.4 Passando props para um componente**

Props são usadas para que os componentes possam se comunicar uns com os outros.

As props aceitam objetos, arrays e funções para serem passadas, semelhante aos atributos. Exemplo de props: **className**, **src**, **alt**, **width** e **height**.

**Exemplo**:



**Adereços são passados ao componente para personalizá-los**.

**Exemplo**:



**Leitura de props dentro de componentes filhos**.

**Exemplo**:

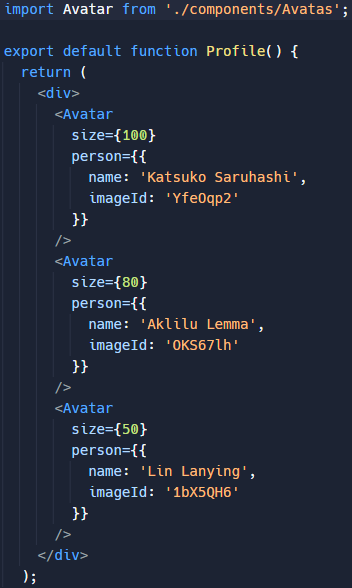
**OBS: props podem ser lidas listando seus nomes separados por virgula dentro ({ prop }) - isso permite o uso dentro de um componente da mesma forma que uma variável**.



**exportação de função filha de Avatar**.



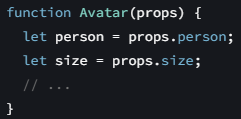
**chamada de componente getImageUrl e criação de componente filho de profile**.



**componente pai que importa avatar e renderiza na tela**.

Os adereços (props) tem a mesma função que os argumentos e as f**unções do componente react só aceitam um único argumento**.

**Exemplo**:



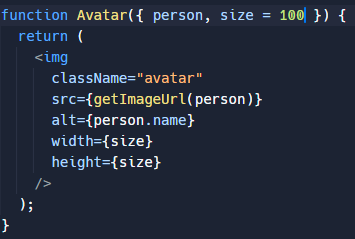
Não há necessidade de uso de todo o props, por esse motivo ele é **desestruturado em adereços individuais**.

sintaxe:

**function nome({ person, size }) { … }**.

Coloque um valor padrão numa prop para retornar caso nenhum valor seja passado, utilize **“=”** para inserir um valor padrão.

**Exemplo**:



**OBS: se passar null ou 0, valor não será usado**.

**Spread deixa a passagem de adereços mais concisas**, ou seja, um componente pai pode passar os adereços para o filho sem listar cada um dos seus nomes (**sintaxe de propagação**).

**Exemplo**:



**aninhamento de tag interna do navegador**:

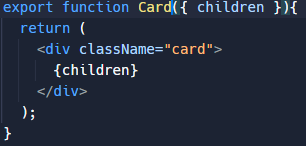
**<div><img /></div>**

**aninhamento de componentes**:

**<Card><Avatar /></Card>**

**Se um conteúdo for aninhado dentro de tag JSX, componente pai recebe conteúdo** (**propriedade children**).

**Exemplo: o componente Card acima está um children prop definido como <Avatar /> e o renderizará**.



**componente Card recebe uma prop filho**.



**Filho Avatar é renderizado dentro do componente pai Card**.

Um componente pode receber props diferentes ao longo do tempo, sendo alguns deles dinâmicos.

* **Props são imutáveis** - o componente pede ao pai para passar props diferentes (novo objeto entrando) descartando seus props antigos que serão posteriormente recuperados na memória tomada por eles.

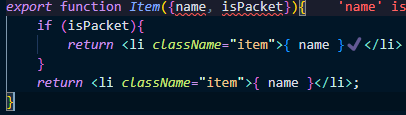
**Props são instantâneos somente leitura no tempo**: cada renderização recebe uma nova versão de props.

**OBS: adereços não podem ser modificados, se necessitar de interatividade define um estado**.

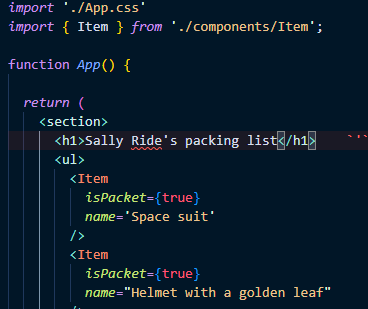
### **1.5 Renderização condicional**

Componentes precisarão exibir diferentes coisas dependendo da condição que lhe foi dada. **O condicional pode ser usado com a sintaxe js como exemplo if, && e ? :**.

**Exemplo**:

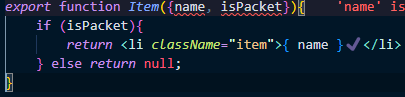


**A marcação só será mostrada se isPacket for true**.



**OBS: fluxo de controle criado pelas condições é tratado pelo JS**.

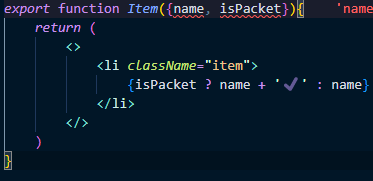
**Exemplo: Retorne null caso deseje não retorna algo**.



**Nesse momento se isPacket receber um valor falso, nada será retorna algo console**.

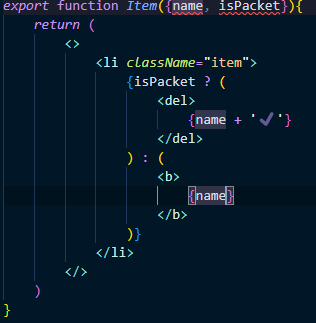
A prática de retornar num componente não boa pois pode trazer problemas quanto ao reconhecimento ao renderizar. **Normalmente, ao escolher incluir ou mostrar um componente filho é importante sempre utilizar o condicional**.

**Exemplo**:



**Uma outra forma de o condicional dentro de um componente seria utilizando o operador ternário. É possível notar que o código se tornou mais compacto e menor**.

**Exemplo: Alinhando tag em componente após o condicional**.

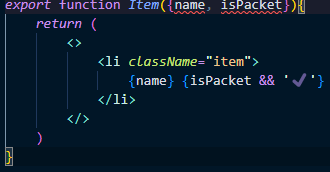


**Se verdadeiro, tag del será inserida dentro li. Caso contrário, a tag b será inserida**.

**O operador && do JS** é normalmente utilizado quando se deseja renderizar algum JSX quando a condição é verdadeira ou não renderizar nada. **Este tipo de condicional normalmente apresenta a condição do lado esquerdo e as resposta no lado direito**.

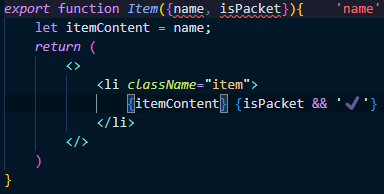
**Obs: lado esquerdo sempre deve dá uma resposta booleana, então evitar colocar apenas um número**.

**Exemplo**:



**Componente renderiza marca de solução se isPacket for true**.

**Exemplo: Atribuindo valores para variáveis em busca de deixar mais simples o código**.

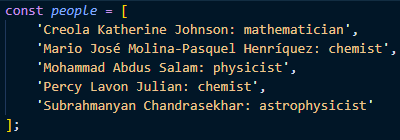


**Valor name atribuido e itemContent**.

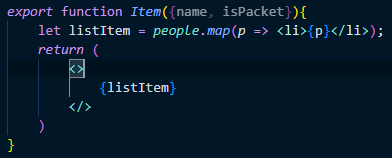
### **1.6 Lista de renderização**

Frequentemente será importante criar várias instâncias do mesmo componente usando dados diferentes ao criar interfaces. **Nessa situação filter e map são utilizados para renderizar lista de componentes.**

**Exemplo**:



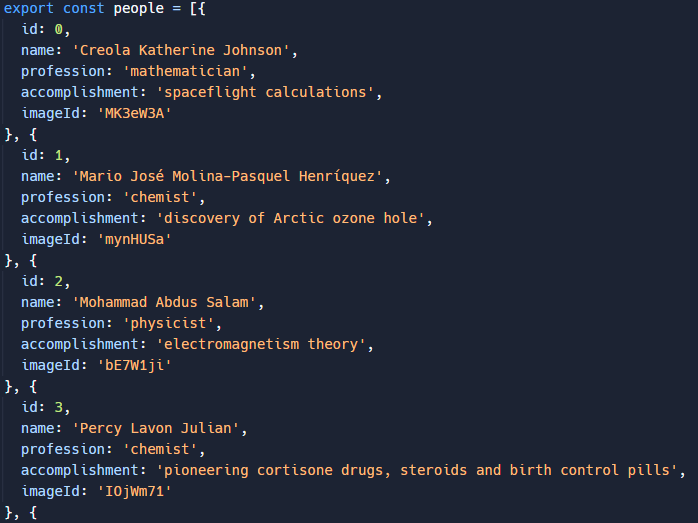
**dados de array a serem mapeados**.



**mapeamento com map**.

**Filter irá servir para filtrar ainda mais os dados** para que somente aqueles que passarem pelo teste serão mostrados dentro do componente.

**Exemplo**:



**array de objetos que possui dados de pessoas que serão consumidos pelo react**.

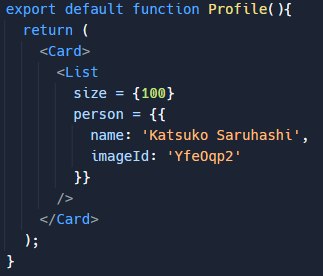
* **elementos JSX dentro de map() precisam de chaves** - indica qual item da matriz cada componente corresponde, ajudando a fazer atualizações na árvore DOM.





**filtragem de pessoas que possuem a profissão chemist**.

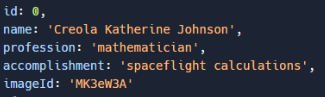




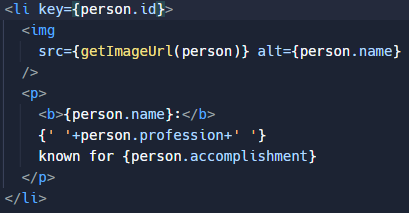
**componente principal renderizando minha lista filtrada**.

**OBS: no caso de arrow function não há necessidade de return se não houver chaves**.

**key** - utilizado para manter ordem em itens dentro de uma lista, pode ser string ou número que o identifica exclusivamente.



**Key chave chamada de id que é representada por um número**.



**chave person.id acrescentada ao <li>**.

**Fragment** - sintaxe longa e explícita que agrupa componentes filhos.



**fragmentos desaparecem do DOM, assim uma lista de h1 e p será produzida**.

**Regras para criar uma chave**:

* chaves únicas entre irmãos (**obs: permitida mesma chave somente para nós em diferentes arrays**).
* devem ser imutáveis.

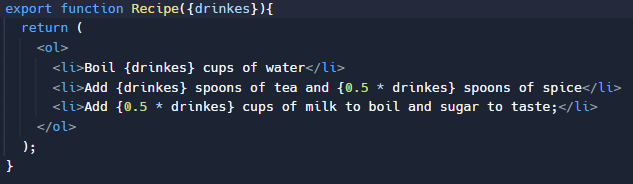
### **1.7 Mantendo os componentes puros**

Componentes puros realizam apenas um cálculo e nada mais. Evita possíveis erros e comportamentos imprevisíveis.

* **características:** sem alterações de objetos e variáveis anteriores ao chamado e dada mesma entrada apresenta a mesma saída.

**OBS: REACT CONSIDERA TODO COMPONENTE CRIADO UMA FUNÇÃO PURA**.

**Exemplo**:



**Criação de um componente puro que sempre irá retornar a mesma saída**.



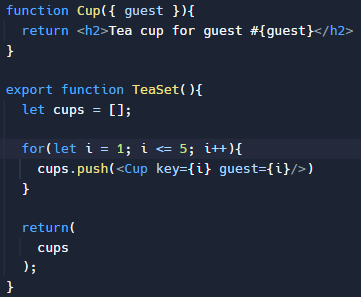
**Após importar Recipe para app, entramos com valores numéricos no componente. Quando drinkes prop são chamados uma segunda vez os valores anteriores se mantêm**.

**OBS: se os componentes renderizados alterarem valores antes da renderização ou valores fora do escopo os mesmos serão considerados impuros**.

**Nota: Mutação - componente altera uma variável pré-existente durante renderização**.

Objetos criados durante renderização não irão afetar um componente puro - **mutação local**.

**Exemplo**:



**Loop cria 5 componentes Cup dentro do componente TeaSet (pai de Cup) que são colocados dentro de um array cups, por fim componente TeaSet é renderizado dentro de App**.

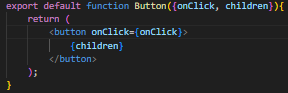
## **2. Adicionando interatividade**

**Manipuladores de eventos** - funções que serão acionadas em resposta às interações do usuário.

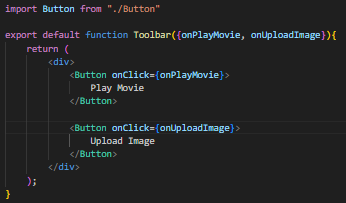
**OBS: componentes integrados <button> suporta apenas eventos de navegador integrados como onClick**.

* Permite criação de próprios componentes

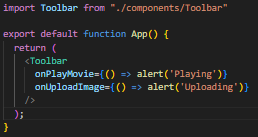
**Exemplo**:



**Criação de button que irá responder a um evento**.



**Componente toolbar com dois props eventos sendo criados**.



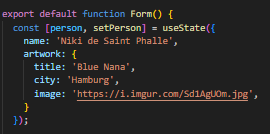
**Componente pai renderizando Toolbar()**.

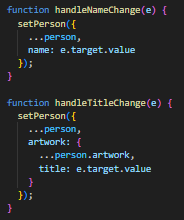
**Estado** - memória específica de cada componente.

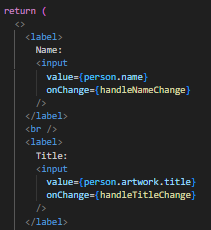
* **useState hook** - adiciona um estado ao componente.
  + **hook** - funções especiais que permitem que seus componentes usem recursos do react.
  + **retornos**: estado inicial e par de valores (**estado atual e função de configuração de estado que permite atualizado**).
* Pode contar qualquer valor, incluindo objetos.

**OBS: objetos e matrizes não devem ser alterados. Copie e crie um novo para atualizar o estado**. Usar “…” para copiar objetos e matrizes.

**Exemplo**:



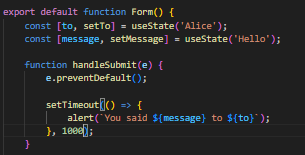


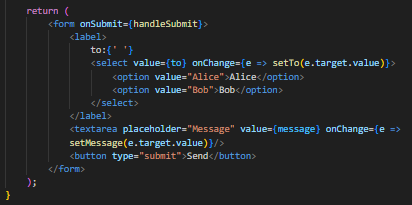


**Processos de renderização**:

* Acionar um render.
* Renderizando o componente.
* Comprometer-se com o DOM.

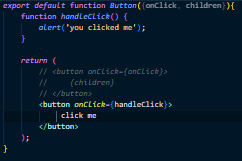
**Exemplo: estado com instantâneo**.





### **2.0 Respondendo a eventos**

**Exemplo: adicionando manipulador de evento**.



**manipulador de eventos:**

* definidos dentro do componente.
* nomes se iniciam com handle seguido do nome do evento.

**Exemplo: definindo manipulador de evento de forma mais concisa**.

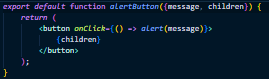


**outra forma**,

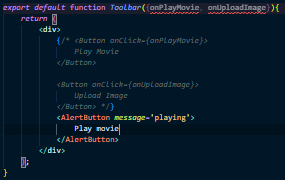


**OBS: FUNÇÕES PASSADAS PARA MANIPULADORES DE EVENTOS NÃO DEVEM SER CHAMADA. EXEMPLO: HANDLECLICK()**.

**Exemplo: lendo adereços em manipuladores de eventos**.



**Criando componente**.



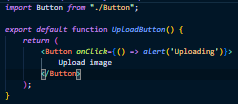
**Lendo o adereço message**.

**Exemplo: passando manipuladores de eventos como props**.

* O componente pai especifica o manipulador de eventos de um filho.
* Chama função passada ao clicar.



C**omponente buttonPlaying passando handle para onclick**.



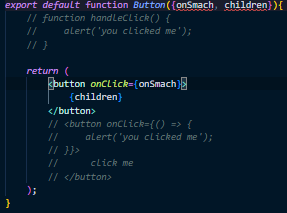
**Componente buttonUpload passando função anônima para onclick**.

**OBS: componentes integrados como button e div suportam apenas nomes de eventos do navegador com onclick.**

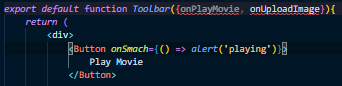
* **No entanto, permite nomear suporte de manipulador de eventos da maneira que quiser para os próprios componentes criados.**

**Props do manipulador de eventos devem começar com on** seguido por uma letra maiúscula.

Exemplo:



**Prop “Smach” de manipulador de eventos**.



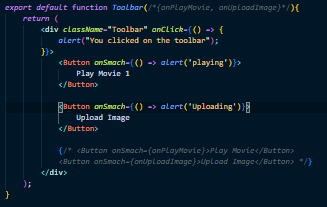
**Utilizando suporte de manipulador de eventos**.

Alterar adereços após interações específicas do aplicativo oferece flexibilidade para alterar como eles são usados posteriormente.

**Exemplo**:



**Exemplo: propagação de eventos em uma árvore.**

****

**Os eventos presentes dentro de cada button borbulham para o componente pai que também terá sua chamada de evento posteriormente.**

**OBS: todos os elementos se propagam no react, exceto onScroll que só funciona na tag JSX**.

Manipuladores de evento **geram o objeto de evento chamado de “e” (evento)**, isso faz com que seja possível fazer a leitura de informações do evento.

* Objeto de evento permite a interrupção do evento **com a utilização de e.stopPropagation()**.

**Exemplo**:



**Dessa forma, a propagação do evento não chega ao pai dos componentes**.

**Efeito de propagação**: **onClick maniulador de button → manipulador chama e.stopPropagation() para impedir borbulhamento && chama onClick função de toolbar → função definida em toolbar exibe alerta do botão**.

**Capture** - pega todos os eventos em um elemento, mesmo que ele tenha interrompido a propagação.

**Exemplo**:



**manipuladores como alternativa à propagação** - permite manipulação de evento pelo componente filho, **ao mesmo tempo que permite que o componente pai especifique algum comportamento adicional**.

* Código pode ser manipulado claramente como resultado de algum evento.

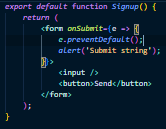
**Exemplo**:



**Um código é executado antes que onClick() propriedade passada pelo pai seja executada**.

**PreventDafult()** - previne que determinado comportamento padrão de algum evento no navegador ocorra.

**Exemplo**:



### **2.1 Estado: Memória de um componente**

Componentes que precisam alterar o estado que está sendo apresentado na tela como resultado de uma interação.

**Variáveis locais** - **não persistem entre as renderizações**. (**ERROS EM ATUALIZAÇÕES DE ESTADO**)

* react renderiza sempre componente do zero.
* react não considera nenhuma alteração nas variáveis locais.
* alterações não acionarão as renderizações.

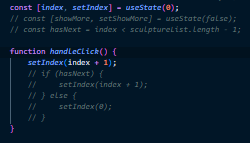
**Requisitos para ocorrer uma atualização de componente com novos dados**:

* retenção de dados entre renderizações.
* acionamento de react para renderização do componente para novos dados.
* **useState**: fornece **variável de estado** para reter dados e **função de configuração de estado** para atualizar a variável e acionar o react.

**Exemplo**:



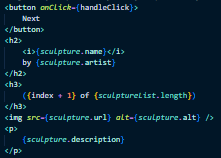
**Importação de useState para variação de estado**.



**Atribuição por desestruturação, sendo possível pegar a variável de estado e a função setter**. **Criação de manipulador de evento handleClick que vai atualizar o componente com novos dados**.



**Variável que vai passar por cada valor presente dentro do objeto**.



**Elementos que são renderizados e que irão sofrer atualização a cada click do botão**.

**OBS: FUNÇÕES QUE COMEÇAM COM “USE” SÃO CHAMADAS DE HOOK**.

* hooks são **funções especiais disponíveis apenas quando o react está renderizando**.
* só podem ser **chamadas no nível superior dos componentes ou de seus próprios ganchos**.
* Não pode usado dentro **condições, loops e funções aninhadas**.

**DICAS**:

* GANCHOS (FUNÇÕES) PODEM SER PENSADOS COMO **DECLARAÇÕES INCONDICIONAIS SOBRE NECESSIDADES DO COMPONENTE**.
* RECURSOS DO REACT DEVE SER **USADO NA PARTE SUPERIOR DOS COMPONENTES**.

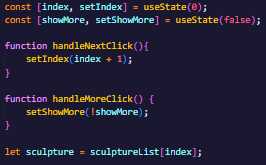
**UseState (!!Lembra de algo!!)**:

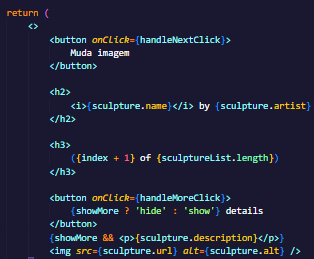
* , guarda informações do index.
* valor inicial de variável (argumento único).
* **Atualizações de estado**: useState sempre recebe a variável de estado armazenada e a função de configuração de estado.

**Nota: por convenção é usado something / setSomething dentro dos colchetes**.

**OBS: uma variável pode ter mais de um estado**.

**Exemplo: estado mudará a cada click**.





**OBS: se variáveis de estado não estiverem relacionadas, crie duas variáveis diferentes**.

**Atualização de estado em hook** ocorre com uma ordem de chamada estável em cada renderização do mesmo componente.

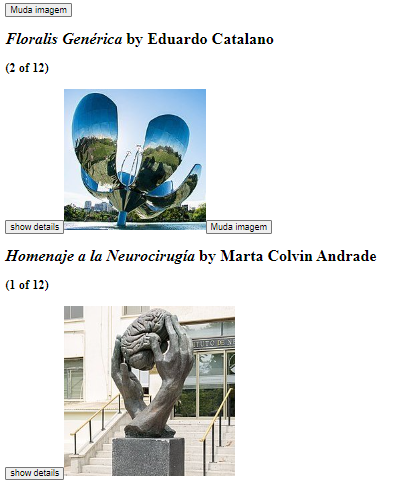
**Estado local** - presente em instância de componente na tela.

* se componente renderiza duas vezes, **cada cópia tem seu estado isolado**.

**Exemplo: isolamento de estado**.



**Renderizando o mesmo componente mais de uma vez**.



**Estado diferentes para os dois componentes (RESULTADO)**.

**OBS: estado é “local” para um local específico na tela**.

* estado totalmente privado para o componente que o declara.

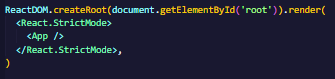
**Nota: para compartilhar o estado entre componentes filhos, basta colocar no componente pai**.

### **2.2 CONTINUAR DAQUI Renderizar e confirmar**

**Renderização inicial** - feita com **CreateRoot** com o nó de destino e seu render método com seu componente.

* **atualização de estado (set função)** do componente enfileira automaticamente uma renderização.
* **appendChild()** - usado para colocar todos os nós DOM criados na tela.

**Exemplo**:



**Renderização subsequente** - chama componente de função cuja atualização de estado acionou a renderização.

* processo recursivo.

**OBS: um componente não deve alterar outros componentes após a sua renderização e sempre deve ocorrer as mesmas saídas para as mesmas entradas**.

**Strict Mode** - react chama função de componente duas vezes para detectar possíveis erros.

**Exemplo**:



**Nota: react apenas altera os nós DOM se houver uma diferença entre as renderizações**.

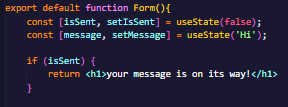
### **2.3 Estado com um instantâneo**

Definir o estado (comportamento instantâneo) **não altera a variável de estado**, mas aciona uma nova renderização.

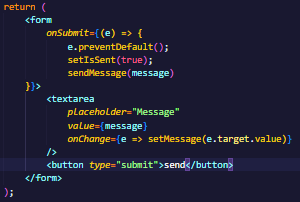
* **interativo** - possui lógica que especifica o que acontece em resposta a uma entrada.

**State** - atualizado quando uma interface reage a um evento.

**exemplo: reagindo a uma mensagem**.



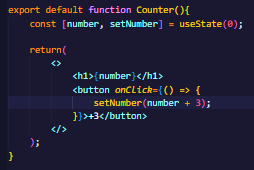
**Criando variáveis de estado e adicionando condicional que irá mandar mensagem se valor de estado for alterado**.



**Mudando valores das variáveis de estado a partir de onSubmit e um click no botão**.

**passos de uma resposta ao evento** - **chama função** novamente → **retorna novo instantâneo** → **react atualiza tela para corresponder ao instantâneo**.

**Exemplo**:



**OBS: o valor de uma variável de estado nunca muda em uma renderização**.

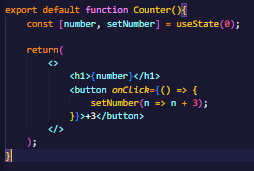
### **2.4 Enfileirando uma série de atualizações de estado**

Definir variáveis de estado enfileira outra renderização.

* **react espera que todo o código de manipuladores de evento** seja executado antes de processar atualizações.
* permite atualizações de mais variáveis de estado.

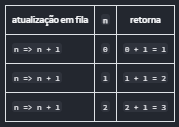
**Lote** - interface de usuário não é atualizada até que manipulador de eventos e código seja concluído.

**Exemplo: atualizando estado várias vezes**.



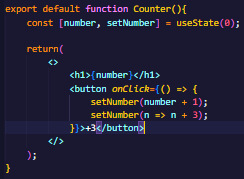
**Passando função que calcula próximo estado com base no anterior (função de atualização)**.

* executada depois de todos os outros códigos no manip. de eventos.
* **salva estado em argumento na função**.



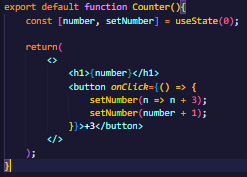
Se o **estado for substituído antes da atualização**, o novo valor obtido será armazenado e atualizado.

**Exemplo**:



**Valor é substituído por 4 e atualizado de 4 em 4**.

Se o **estado for substituído depois da atualização**, o mesmo irá manter o seu novo valor sem considerar a atualização.



**OBS: funções do atualizador devem ser puras e retornar apenas o resultado**.

**Info: o argumento da função de estado sempre deve ter as letras da variável de estado**.

### **2.5 Atualizando objetos no estado**

Estado pode conter qualquer valor javaScript, incluindo objetos.

**Exemplo**:



**Estado recebendo valor numérico. Os valores também podem ser booleanos, strings, dentro outros**.

**OBS: para atualizar um objeto é importante criar um novo ou fazer a cópia de um existente**.

**mutação** - possibilidade de alterar o conteúdo de alguma coisa, tipo um objeto.

**Exemplo**:

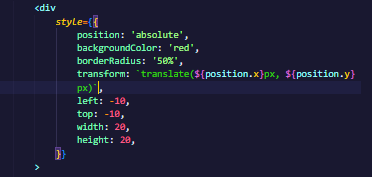


* **devem ser tratados como imutáveis (somente leitura), ou seja, sempre buscar substituídos**.

**Exemplo: objeto no estado**.

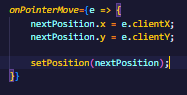


**Estado contendo objeto com propriedade x e y que é modificada pela função de configuração de estado**.



**retorna estado com novos valores de posição**.

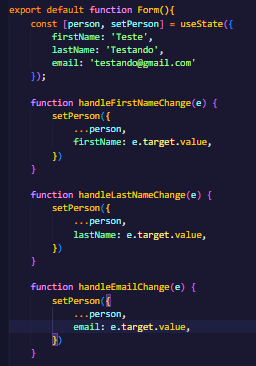
**Mutação local** - Uma outra forma de atualizar o objeto dentro da função de atualização:



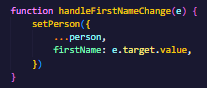
**Cria novo objeto e retorna na função**.

**OBS: não alterar objetos existentes que já estão no estado**.

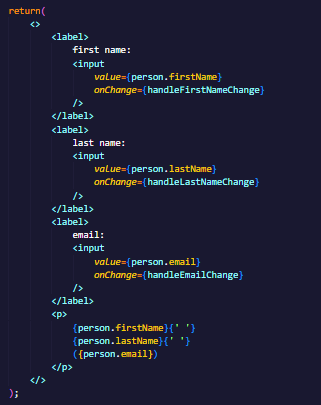
**Exemplo: copiando valores existentes para um novo objeto**.



**Alterando os valores dentro do objeto passado para função atualizadora**.



**Função altera objeto e salva os valores já existentes**.

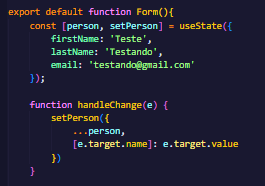


**Retornando componentes com novos valores atualizados**.

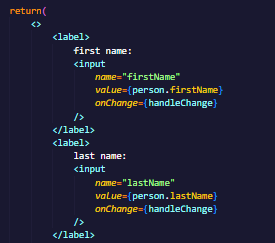
**OBS: os 3 pontos antes do objeto person é usado para pegar todos os valores existentes, ajudando a copiar todas as propriedades**.

* **sintaxe: …person**.
* Para atualizar uma propriedade aninhada é necessário utilizar a mesma mais uma vez.

**Exemplo: Uma outra forma de atualizar os campos com apenas uma manipulador de eventos**.

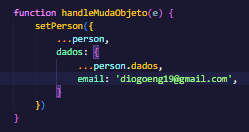


**e.target.name está se referindo ao name de input**.



**OBS: estados devem ser imutáveis**.

**Exemplo: atualizando estado de objeto aninhado**.



**Valores de email são atualizados para um novo estado. Dessa forma, os valores de anteriores de person e dos dados são pegos, e o email é atualizado**.

**Nota: objetos não estão aninhados, apenas apontando para outros objetos**.

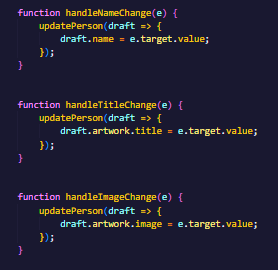
**Spreads aninhados (Immer)** → biblioteca que permite escrever com sintaxe mais conveniente.

* não sobrescreve o estado passado.
* **draft (tipo especial de objeto - Proxy)**: registra todas as modificações realizadas com ele e produz um novo objeto com as novas modificações.
* instalação: **npm i use-immer**.

**Exemplo**:



**importa immer e inicia estados definidos como objetos**.



**Função de atualização de estado utilizando immer**.



**Componentes com manipuladores de estado usando immer**.

**Benefícios da imutação**:

* **depuração** - clara visão das mudanças de estado.
* **otimizações** - verificação de alteração mais simples.
* **novos recursos** - implementação de novos recusos.
* **Alterações de requisitos** - reutilização de valores anteriores que ficam presentes na memória.
* **implementações mais simples**.

### **2.6 Atualizando matrizes no estado**

Arrays devem ser tratados como **imutáveis** assim como os objetos em react.

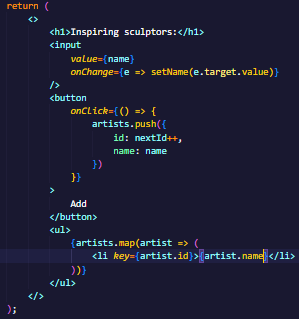
* Um novo array deve ser criado ou copiado para mudar o estado.
* Novo array passado para função de configuração de estado.

**OBS: métodos que não podem ser usados**.

* **push**, **unshift**. (adiciona)
* **pop**, **shift**, **splice**. (remove)
* **splice**, **arr[i] = atribuição**. (substitui)
* **reverse**, **sort**. (ordena)

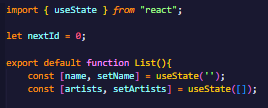
**Nota: slice (copia array ou parte dele) e splice (transforma o array, inserindo ou excluindo itens)**.

**Exemplo**:



**A atualização de estado não ocorre porque push() está transformando um array**.

**Exemplo: modo correto para resolver o problema**.



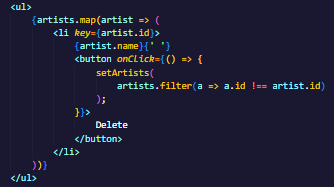
**os estados e suas funções de configuração sendo criadas**.



**o retorno de marcações html onde button irá atualizar o estado com a entrada de um novo array**.

**OBS: a sintaxe de distribuição pode ser inserido no início e no final de um novo item**.

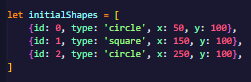
**Exemplo: removendo itens com filtragem**.



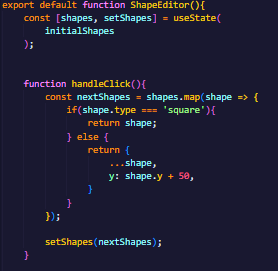
**marcação html onde um filtro está passando pelos valores do array e eliminando aqueles que não passam pelo condição, por fim gerando um novo array**.

**map()** - pode ser usada para alterar alguns ou todos os valores do array.

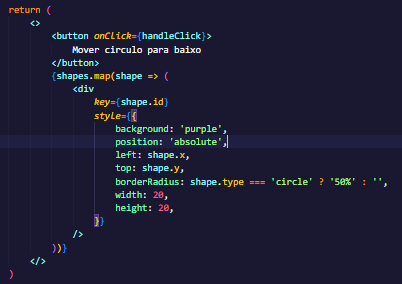
**Exemplo: criando um novo array com map()**.



**estado inicial do array**.



**função manipuladora de estado que estará mudando o estado a cada click no botão**.

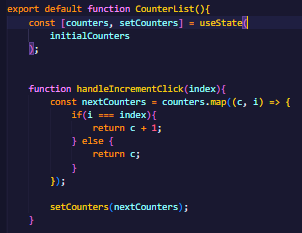


**tad div que possui props que irão determinar se o elemento será um quadrado ou um circulo**.

**Exemplo: substituindo itens com map**.



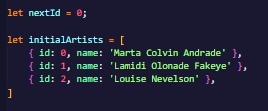
**estado inicial do elemento**.



**manipulador de evento que mudará o valor do array**.



**Exemplo: inserindo valores em um array com a utilização de slice() e a sintaxe de distribuição**.

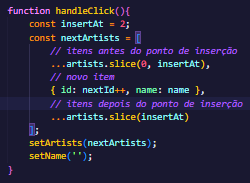


**variável que vai servir como index para mudar valor e estados iniciais**.

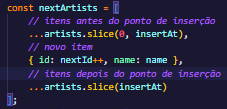


**estados sendo iniciados e função manipuladora de estado que terá o objetivo de inserir um novo valor entre os pontos fatiados**.

**Exemplo: fazendo alterações em um array**.



**A cópia da matriz original vai para nextArtists que poderá ser modificada e posteriormente passada a função de configuração de estado**.



**dessa forma é possível utilizar métodos de mutação sem afetar o estado**.

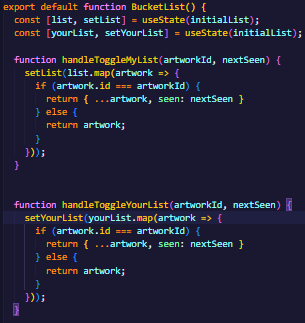
**OBS: mesmo copiando uma matriz os seus itens existentes não poderão ser modificados pois as cópias são superficiais**.

**Nota: para solucionar o problema de mutação citado acima, basta utilizar a idéia de atualização de objetos aninhados**.

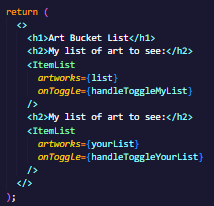
* cópia de itens individuais que deseja alterar.

**OBS: O objeto nada mais são que referências apontadas pela matriz, ou seja, eles não estão localizados dentro da matriz**.

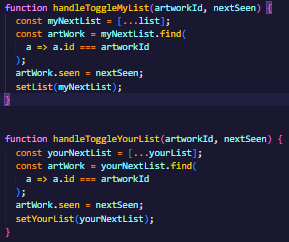
**Exemplo: isolando estados**.



**manipuladores de evento utilizam map para substituir um item antigo por uma versão atualizada sem mutação**.



**outro modo onde ocorre erro**,

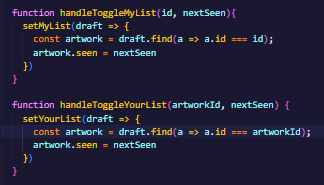


**Este código apresenta erro porque sua variável de estado está sendo compartilhada e não se encontra isolada**.

**OBS: Somente objetos que foram acabados de criar podem ser alterados**.

**Atualização com Immer** - reestruturação de estado mais concisa e sintaxe mais concisa.

**Exemplo: funções do componente bucketList com Immer**.



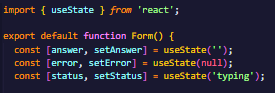
**com isso a mutação não afeta o estado, pois o valor alterado é da draft objeto especial fornecido pelo Immer**.

**Nota: Immer constrói próximo estado do zero de acordo com as mudanças no draft**.

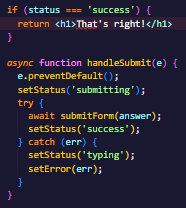
## **3. Gerenciando estados**

**Descreve a interface de usuário para os diferentes estados visuais do componente**, acionando as alterações visuais de estado em resposta à entrada do usuário.

**Exemplo: questionário simples**.



**Variáveis de estados criadas**.



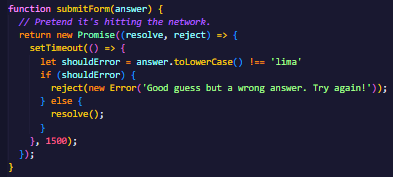
**condição será executada se status for sucesso. função assíncrona que seta estado para submitting e espera resposta da função submitForm**. **Erro lançado se resposta for errada e estado setado para início**.



**pega valor de input**.



**marcações html retornadas**.



**função que irá lançar uma promesa dependendo se resposta for correta**.

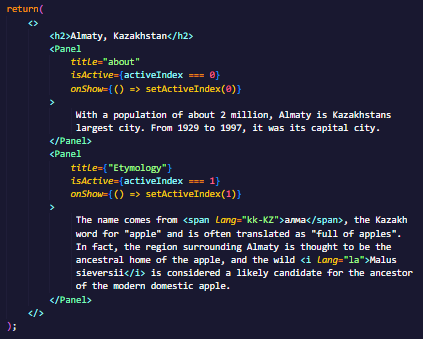
**OBS: o estado não pode conter informações redundantes ao complicadas ou duplicadas**.

**Compartilhamento de estado** → ocorre por meio de um componente pai que passa o estado por meio de props para os seus filhos (**aumento de estado**).

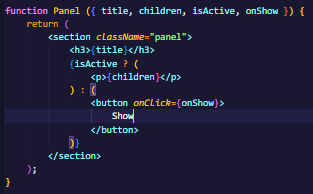
**Exemplo: componente pai mantendo estado e especificando props para os filhos**.



**definindo estados**.



**o componente filho Painel recebe os estados e a função manipuladora de estado do seu pai**.



**componente filho painel**.

react precisa definir quais partes da árvore atualizar / manter e quais devem ser descartadas / recriadas, para isso a **utilização de key serve para forçar um componente a redefinir seu estado sem afetar os demais na árvore**.

* informa que o componente é diferente ao react.

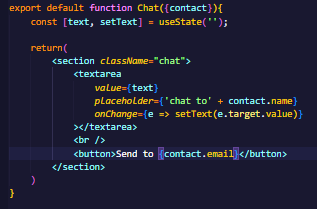
**Exemplo: redefinindo estados com key**.



**Componente message que terá cada chat redefinido de acordo com sua chave**.



**lista de contatos que serão definidos**.

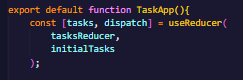


**chat que será definido a partir do contato selecionado**.

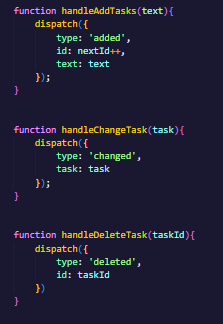
**Função reducer** - consolida lógica de atualização de estado fora do componente em uma única função.

* manipuladores de eventos mais concisos (**especificam ações de user**).

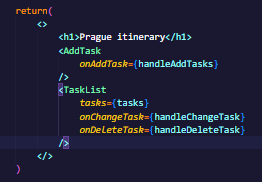
**Exemplo: especificando estado com função reducer**.



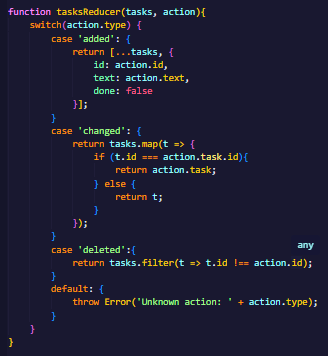
**função reducer recebendo um switch de manipulação de estado e os valores iniciais**.



**funções manipuladoras de evento**.



**marcações retornadas de TaskApp**.



**função com switch que retorna manipuladores de estado dependendo do case selecionado**.



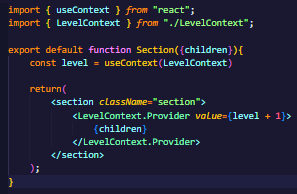
**valores para o estado inicial**.

**Context** - permite que o componente pai disponibilize algumas informações para qualquer componente na árvore abaixo dele (props passadas implicitamente).

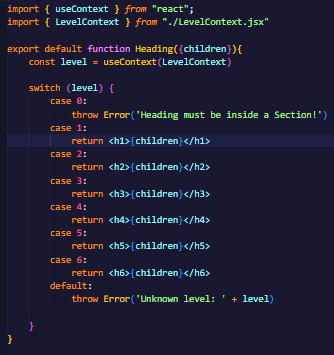
**Exemplo: passando o contexto para componentes filhos**.



**variável levelContext apenas está criando um contexto**.



**o componente Section está recebendo um filho e repassando o contexto criado por levelContext**.



**o componente Heading está recebendo o seu contexto do seu pai Section que irá determinar o tipo de heading a ser utilizada**.

**Redutores** - permitem consolidar a lógica de atualização de estado de um componente.

* gerenciamento de estado de uma tela mais complexa.
* **permite despacho por meio de outros componentes para atualização** de estado.

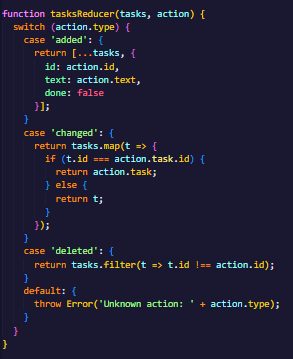
**Exemplo: criando um redutor**.



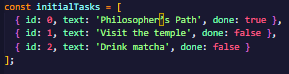
**criação de contexto inicial**.



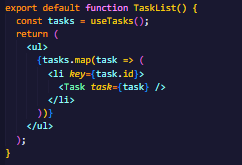
**componente que possui o contexto e o reducer com os estados iniciais**.



**função reducer com a lógica de atualização de estado**.



**contextos iniciais**.



**componente task**.



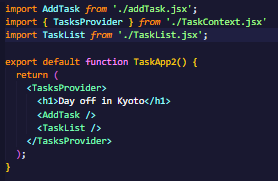
**componentes task com todas as lógicas e com o contexto repassado do seu pai**.



**retorna de Task**.



**componente está obtendo o contexto de seu pai**.



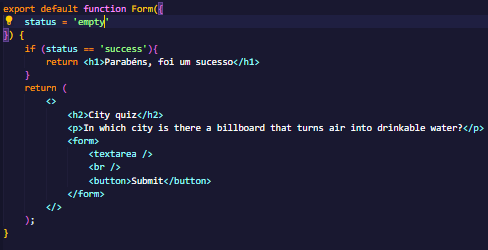
**componente pai com os seus filhos**.

### **3.0 Reagindo à entrada do estado**

Diferentes estados podem ser obtidos pelo componente em resposta à entrada do usuário.

* **programação imperativa** - descreve estados de resposta ao usuário (**identifica** estados visuais, **determina** quem determina mudança, **representa** estado na memória, **remove** variáveis de estado não essenciais e **conecta** manipuladores de eventos).

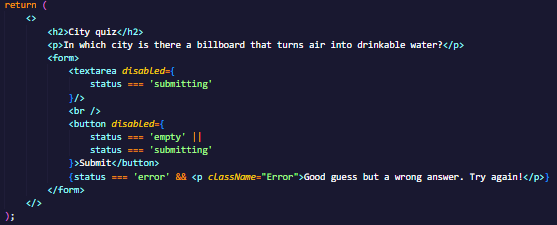
**Exemplo: descrevendo estados**.



**inicia com um estado inicial “empty”, que pode ser atualizado se status for igual a success**.

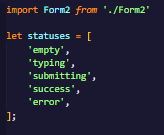
* interação rápida na interface do user sem lógica.

**Exemplo: melhorando o código**.

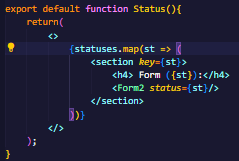


**as tags serão desabilitadas dependendo do estado que se encontrem, e um valor de error será lançado ao final do formulário para indicar uma situação de possível error**.

**Exemplo: componente com mais de um estado visual**.



**todos os estados são colocados dentro de um array**.



**o componente status está mapeando os estados do array e retornando uma sessão com o componente form2 como filho que recebe cada estado**.

### **Etapa 2: determinar o que desencadeia…** [**https://react.dev/learn/reacting-to-input-with-state**](https://react.dev/learn/reacting-to-input-with-state)

## **Framework**

### **Next.js**

**Framework react full-stack** - versátil e permite criar aplicativos react de qualquer tamanho.

Mantido pela Vercel

**App router** - redesenho de APIs do next.js. Permite busca de dados em componentes assíncronos que rodam no servidor ou durante build.

**Benefícios:**

* implantação em node.js
* hospedagem sem servidor ou com servidor
* podem ser implantados em hospedagem estáticas

### **Remix**

**Framework React full-stack com roteamento aninhado** - permite criação de aplicativo em partes aninhadas, carregamento de dados em paralelo e atualização de resposta às ações do usuário.

Mantido pelo Shopify.

**Benefícios:**

* Implantação em nodo.js
* hospedagem sem servidor ou gravando um adaptador

### **Gatsby**

**estrutura React para sites rápidos baseados em CMS (sistema de gestão de conteúdos)** - apresenta diversos plugins e camada de dados graphQL.

Mantido pela Netlify.

**Benefícios:**

* integração de conteúdo mais simples
* APIs e serviços em site
* implementação de site totalmente estático em hospedagem estática

### **Expo**

**Estrutura React para criação de aplicativos android, IOS e WEB com UIs nativas** - fornece SDK para react native

Mantido pela Expo.

**Benefícios:**

* criação de app é gratuita
* envia para lojas de app sem restrições
* ofereço serviços de nuvem opcionais