Use template

~~yarn create react-app moveit --template=typescript~~

yarn start

yarn create next-app moveit-next

yarn add typescript @types/react @types/react-dom @types/node -D

yarn dev

Next.js

É um framework do react com estruturas prontas, ex: roteamento

Um tipo de camada adicional entre o backend e o frontend

SPA – Single Page Application.

Troca de rotas muda o conteúdo, mas não a página

O problema disso é a indexização por motores de busca

SSR – Server Side Rendering

SSG – Static Site Generation

Evita dezenas de requisições semelhantes em uma página, ex: G1 é atualizado

1x a cada 15 min, ao invés de fazer milhares de requisições no DB para cada

user, faz uma e salva isso.

Criar projeto

Usando create-react app

Só tem SPA, não tem SSR e SSG

Gatsby

Framework para SSG

Next.js

Tem SPA, SSR, SSG

Código index.tsx

Ao renderizar o DOM, busque o elemento ROOT e dentro dele, jogue o que está App

O App é um componente do React

ReactDOM.render(

  <App />,

  document.getElementById('root')

);

**Componente React**

É uma função que retorna JSX

Permite reaproveitamento de código

Ex: Button()

Obs: Não se pode ter vários componentes em uma função, se precisar colocar 3 botões, coloque uma DIV englobando os 3.

function App() {

  return (

    <h1>

      Hello NLW

    </h1>

  );

}

export default App;

**Propriedades React**

Semelhante a Atributos do HTML

Componente:

interface ButtonProps {

  color: string;

}

export function Button(props: ButtonProps){

  return(

    <button type="button">

      Botão

      <strong>Teste</strong>

    </button>

    )

}

App:

function App() {

  return (

    <div>

      <Button color="red" />

      <Button color="green" />

      <Button color="blue" />

    </div>

  );

}

**Propriedade Children**

Sempre que passamos algo dentro de um componente, ele é children.

Ex: Botão 1 = children

function App() {

  return (

    <div>

      <Button color="red">

        Botão 1

      </Button>

    </div>

  );

}

**Estado**

Armazenar informações que vão variar, ao longo do tempo.

Não são variáveis simples do JS.

import { useState } from 'react';

export function Button(props: ButtonProps){

  const counter = useState(1)

…

}

O useState não retorna apenas a variável em si, mas a variável e uma função para atualizar o valor do conter, então é possível desestruturar

export function Button(props: ButtonProps){

  const [counter, setCounter] = useState(1)

…

}

OBS: O React utiliza IMUTABILIDADE, ele não edita uma variável existente, ele cria um valor novo.

  // Da erro

  function increment() {

    counter++;

  }

  // Ok

  function increment() {

    setCounter(counter + 1);

  }

Dessa forma é possível contar o número de clicks em cada botão de forma isolada, pois o estado é definido dentro de cada componente

import { useState } from 'react';

interface ButtonProps {

  color: string;

  children: string;

}

export function Button(props: ButtonProps){

  const [counter, setCounter] = useState(1)

  function increment() {

    setCounter(counter + 1);

  }

  return(

    <button

      type="button"

      style={{backgroundColor: props.color}}

      onClick={increment}

    >

      {props.children} <strong>{counter}</strong>

    </button>

    )

}

**Incluir código JS dentro do HTML**

Usar chaves

  return(

    <button type="button">

      {props.color}

    </button>

    )

**Incluir objeto de JS**

O objeto também usa chaves

Quando você quer utilizar o objeto dentro do HTML, usar 2 chaves

    <button type="button" style={{backgroundColor: props.color}}>

      Botão

      <strong>Teste</strong>

    </button>

**CSS**

**Um único level abaixo de uma class**

Quando quero estilizar um único level (evita div > div > div), coloco:

.class > div { ... }

    return(

        <header className="experience-bar">

            <span>0 xp</span>

            <div></div>

            <span>600 xp</span>

        </header>

    );

  .experience-bar > div {

    flex: 1;

    height: 4px;

    border-radius: 4px;

    background: var(--gray-line);

    margin: 0 1.5rem;

    position: relative;

  }

**Utilizar o centro como referência em um elemento com position Absolute**

Translate no X de 50%

  span.current-experience {

    transform: translateX(-50%);

  }

**useEffect**

Um tipo de interrupção

Executar a função () => {} sempre que o [] mudar

    useEffect(() => {

        if (active && time > 0) {

            setTimeout(() => {

                setTime(time - 1);

            }, 1000);

        } else {

            setActive(false);

            setTime(25 \* 60);

        }

    }, [active, time]);