REACT COM TYPESCRIPT

Aprenda esse feitiço

Luísa Foppa

01

INTRODUÇÃO AO REACT

Aqui usaremos o npm, gerenciador de pacotes do NodeJS. Para inicializar um projeto React com typescript execute o seguinte comando no terminal:

```
npx create-react-app my-app --template typescript
```

Com React, podemos transformar até mesmo as páginas mais estáticas em experiências interativas e encantadoras. Após as instalações, vamos começar com um simples feitiço :

```
import React from 'react';

const App: React.FC = () => {
  return <div>Hello, wizarding world!</div>;
};

export default App;
```

Nele importamos a biblioteca React e criamos um componente funcional chamado App. Dentro deste componente, retornamos um elemento contendo a mensagem "Hello, wizarding world!“, por fim fazemos a exportação desse componente

02

ENCANTAMENTOS DO TYPESCRIPT

Agora que você começou a explorar o mundo de React, é hora de adicionar um toque de magia extra com TypeScript. Esta é uma linguagem que estende o JavaScript adicionando tipos estáticos, proporcionando mais segurança e clareza ao desenvolvimento de aplicativos. :

Vamos dar uma olhada em como os encantamentos de TypeScript podem aprimorar nosso feitiço inicial

```
const sayHello = (name: string): string ⇒ {
  return `Hello, ${name}!`;
};

console.log(sayHello('Harry'));
// Output: Hello, Harry!
```

Neste trecho de código, declaramos uma arrow function chamada **sayHello** que espera receber um parâmetro **name** do tipo **string.**

Essa tipagem mágica facilita a legibilidade do código e a detecção de erros em tempo de compilação.

Graças aos encantamentos de TypeScript, nosso código se torna mais claro e menos propenso a erros.

Algum dos tipos de variáveis mais comuns são:

1. **string:** dados de texto.
2. **number:** números, tanto inteiros quanto de ponto flutuante.
3. **boolean:** valores verdadeiro/falso.
4. **array:** uma coleção de elementos do mesmo tipo.
5. **enum:** um conjunto de constantes nomeadas.
6. **any:** qualquer tipo de dado (geralmente usado quando o tipo não é conhecido antecipadamente ou quando se deseja desabilitar a verificação de tipos.)
7. **void:** ausência de valor
8. **null e undefined:** valores nulos ou indefinidos.
9. **object:** representa um tipo de objeto

```
let feiticos: string[] = ["Lumos", "Accio"];
enum Casa {
  Grifinoria
  Sonserina,
  Corvinal,
  Lufa-Lufa
}
let casa: Casa = Casa.Grifinoria;
let personagem: object = {
  nome: "Hermione Granger",
  idade: 11 };
```

Exemplo de declaração de variáveis sendo respectivamente : array, enum e object:

03

ELEVE SUA MAGIA COM COMPONENTES

Componentes são os blocos de construção fundamentais de qualquer aplicativo.

Eles nos permitem dividir a interface do usuário em partes reutilizáveis e independentes, facilitando a manutenção e a organização do código.

Neste feitiço temos o componente **Greeting**, que aceita a propriedade **characterName** do tipo **string**. Com esse poder, podemos conjurar interfaces de usuário poderosas e flexíveis.

```tsx
import React from 'react';

// Definindo a interface para as propriedades
//do componente
interface GreetingProps {
  characterName: string;  }

// Criando um componente funcional de saudação
const Greeting: React.FC<GreetingProps> =
({ characterName }) => {
  return <div>Hello, {characterName}!</div>;
};

// Utilizando o componente Greeting
const App: React.FC = () => {
  return (
    <div>
      <Greeting characterName="Harry" />
      <Greeting characterName="Hermione" />
      <Greeting characterName="Ron" />
    </div>
  );
};

export default App;
```

04

GERENCIAR ESTADOS E PROPS

Assim como os bruxos precisam dominar o uso de varinhas mágicas, nós precisamos dominar o gerenciamento de estados e props.

Os estados representam dados que podem mudar ao longo do tempo, enquanto as props são propriedades que são passadas para os componentes

```tsx
import React, { useState } from 'react';

interface SpellCounterProps {
  initialValue: number;
}

// Criando um componente funcional SpellCounter
const SpellCounter: React.FC<SpellCounterProps> =
({ initialValue }) => {
  // Definindo o estado inicial usando useState
  const [count, setCount] = useState<number>(initialValue);

  // Magia para incrementar o contador
  const incrementar = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Contagem: {count}</p>
      <button onClick={incrementar}>Incrementar</button>
    </div>
  );
};

// Utilizando o SpellCounter com um valor inicial
const SpellApp: React.FC = () => {
  return <SpellCounter initialValue={0} />;
};

export default SpellApp;
```

05

ENCANTAMENTOS AVANÇADOS COM HOOKS

Hooks são feitiços avançados que auxiliam na funcionalidade dos componentes.
O componente ViraTempo exibe um contador que começa em 0 e aumenta a cada segundo.

Com o encantamento useEffect, lançamos um feitiço que o atualiza continuamente

Este encantamento nos permite acompanhar o fluxo do tempo, tornando nossa aplicação reativa e dinâmica.

```tsx
import React, { useState, useEffect } from 'react';

// Criando um componente funcional ViraTempo
const ViraTempo: React.FC = () => {
  // Definindo o estado inicial para contar os segundos
  const [seconds, setSeconds] = useState<number>(0);

  // Encantamento useEffect para atualizar
  //a cada segundo
  useEffect(() => {
    const viraTempo = setInterval(() => {
      setSeconds(prevSeconds => prevSeconds + 1);
    }, 1000);

    // Desfazendo o encantamento
    return () => clearInterval(viraTempo);
  }, []);

  return <div>Segundos: {seconds}</div>;
};

// Utilizando o ViraTempo
const App: React.FC = () => {
  return <ViraTempo />;
};

export default App;
```

O objetivo desses encantamentos é dar acesso a recursos avançados, como estado interno e efeitos secundários, sem precisar recorrer às complexidades das classes.

Os mais utilizados são **useState** e **useEffect,** como na magia anterior.

O useEffect é ativado sempre que algo muda no ambiente do componente ,como a atualização de uma variável ou a montagem/desmontagem de um componente

```
useEffect(() => {
  // Define um temporizador para simular a duração da poção
  const timer = setTimeout(() => {
    setPoção('Poção de Invisibilidade');
  }, 5000); // A poção é atualizada após 5 segundos

  // Limpa o temporizador quando o componente é desmontado
  return () => clearTimeout(timer);
}, [poção]); // Executa o efeito sempre que a poção muda
```

Aqui o useEffect agora recebe [poção] como segundo argumento, o que significa que ele será executado sempre que a variável poção mudar. Dessa forma, quando a poção for atualizada para "Poção de Invisibilidade" após 5 segundos, o useEffect será acionado novamente, mas como a poção não mudará mais, o temporizador não será redefinido. Isso garante que a atualização da poção ocorra apenas uma vez, após 5 segundos, mas o useEffect continua observando mudanças na poção para possíveis futuras atualizações.

OBRIGADA POR LER ESSE EBOOK!

Espero que tenha gostado

Esse ebook foi gerado por IA e a diagramação por humana. Ele é um projeto com base nas orientações do professor Felipe Aguiar na plataforma da Dio

Mais informações presentes no repositório desse projeto: