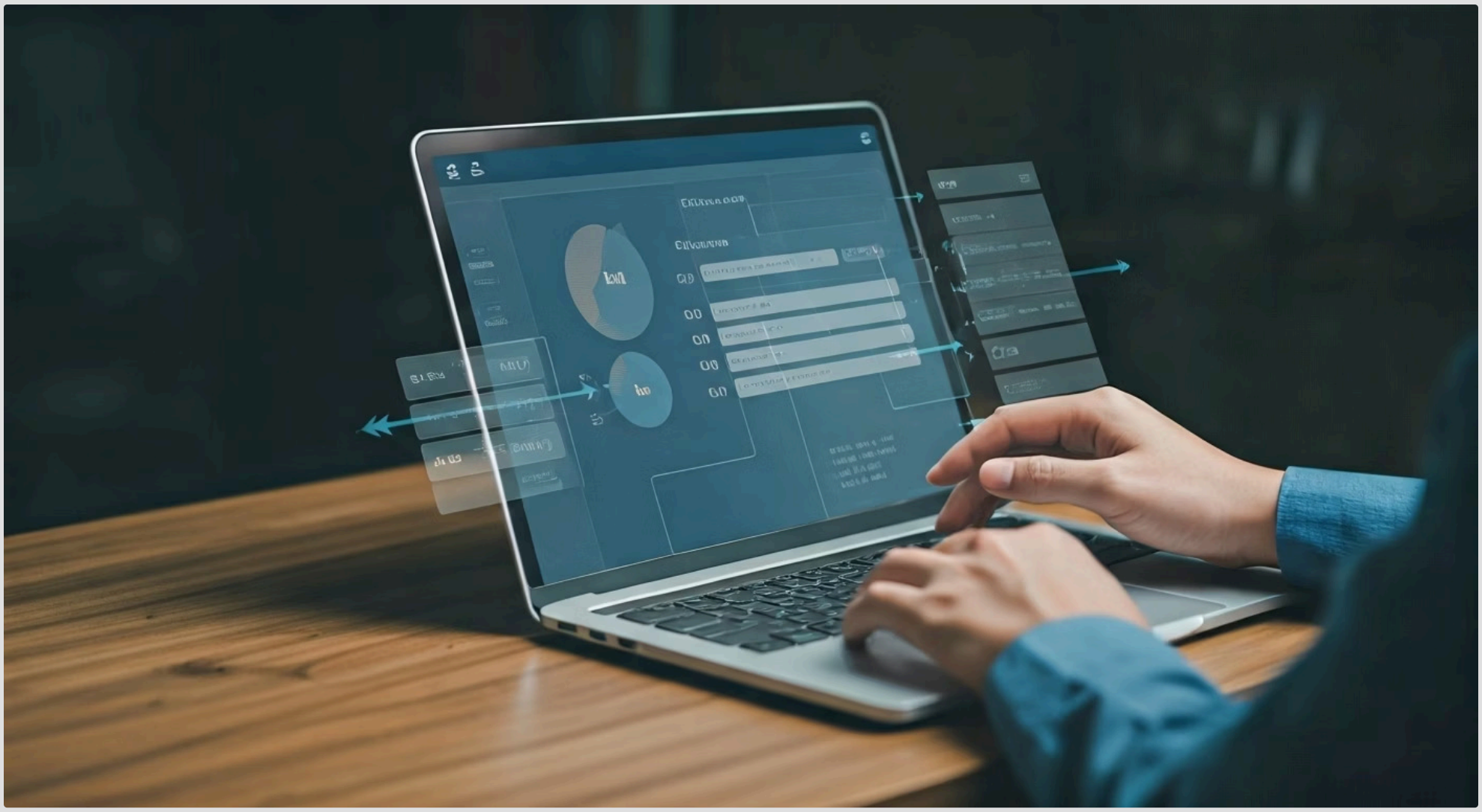
Aula 24 – React: Eventos e Formulários

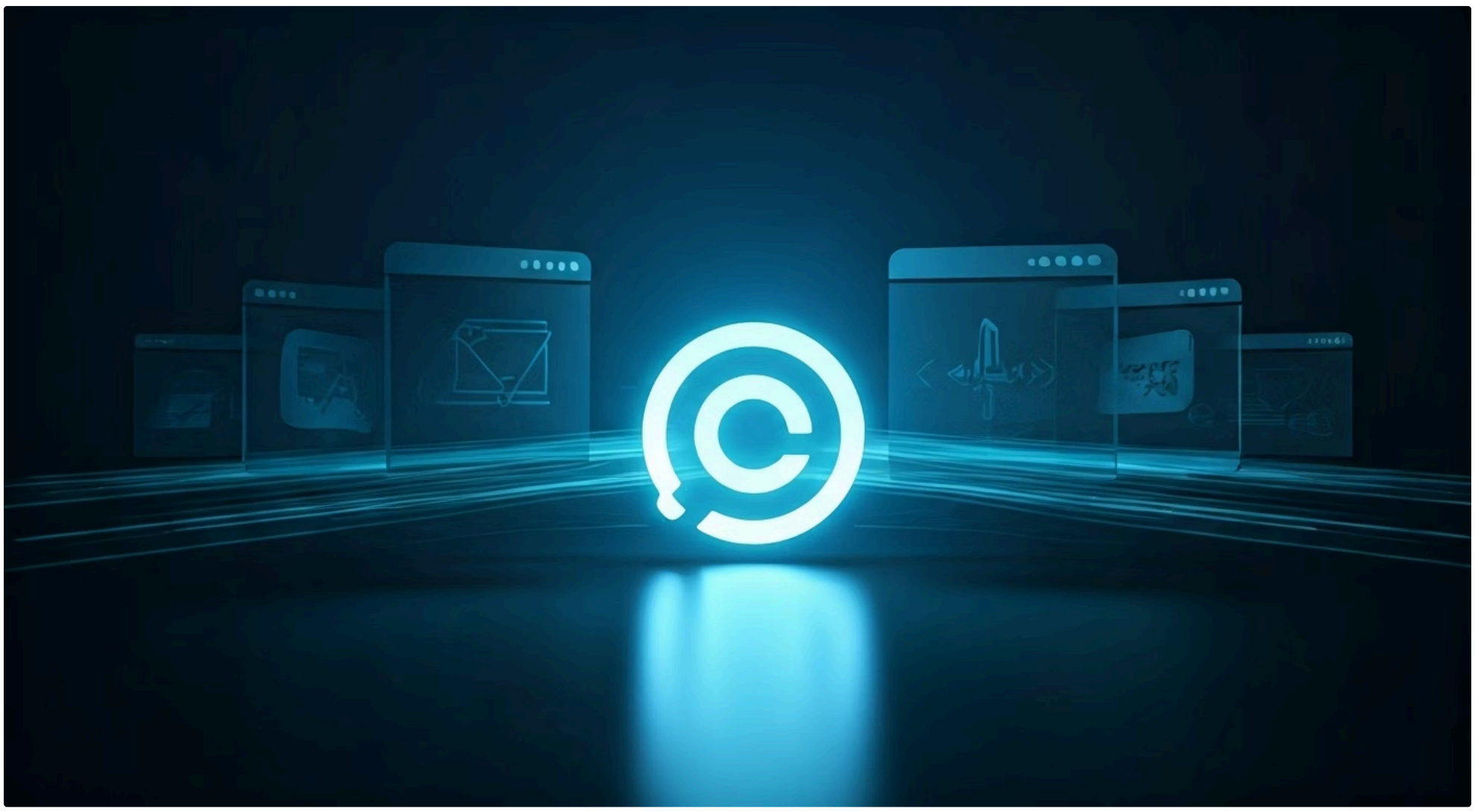


Bem-vindos à jornada de construção de interfaces interativas e dinâmicas com React! No mundo do desenvolvimento web, a capacidade de uma aplicação responder às ações do usuário é o que a torna viva e útil. Pense em cada clique, cada digitação, cada seleção como uma conversa entre o usuário e o sistema. É através dessa interação que os dados são coletados, as funcionalidades são acionadas e a experiência se torna significativa.

Nesta aula, mergulharemos no coração dessa interação: os eventos e os formulários em React. Você já deve ter uma base sólida de como eventos funcionam no JavaScript puro, mas o React traz sua própria abordagem, mais declarativa e eficiente, para gerenciar essas interações. Compreender essa filosofia é crucial para construir aplicações robustas, fáceis de manter e que ofereçam uma experiência de usuário fluida.

Nosso objetivo principal é equipar você com o conhecimento e as ferramentas para manipular eventos de forma eficaz e construir formulários complexos e acessíveis. Ao final, você será capaz de criar componentes que reagem inteligentemente às entradas do usuário, gerenciar o estado de formulários de maneira organizada e aplicar as melhores práticas que garantem não apenas funcionalidade, mas também acessibilidade e performance, pilares do desenvolvimento moderno. Prepare-se para transformar suas interfaces estáticas em experiências interativas e responsivas!

A Essência dos Eventos em React: O Tradutor Universal



No universo do desenvolvimento web, a interação do usuário é o motor que impulsiona a experiência. Cada clique em um botão, cada caractere digitado em um campo de texto ou cada item selecionado em um menu suspenso gera um "evento". No JavaScript tradicional, lidamos com esses eventos diretamente, adicionando ouvintes a elementos do DOM. No entanto, o React, com sua abordagem declarativa e seu DOM virtual, introduz uma camada de abstração que simplifica e padroniza essa interação.

> Imagine que você está em uma conferência internacional, e cada participante fala um idioma diferente. Para que todos se entendam, é necessário um tradutor universal.

Em React, os eventos funcionam de maneira semelhante. Em vez de lidar diretamente com os eventos nativos do navegador (que podem ter pequenas diferenças entre eles), o React envolve esses eventos em um objeto chamado **SyntheticEvent**. Este objeto é um "tradutor universal" que garante que seus eventos se comportem de forma consistente em todos os navegadores, oferecendo uma API unificada e familiar.

Compatibilidade

Resolve diferenças entre navegadores automaticamente

Performance

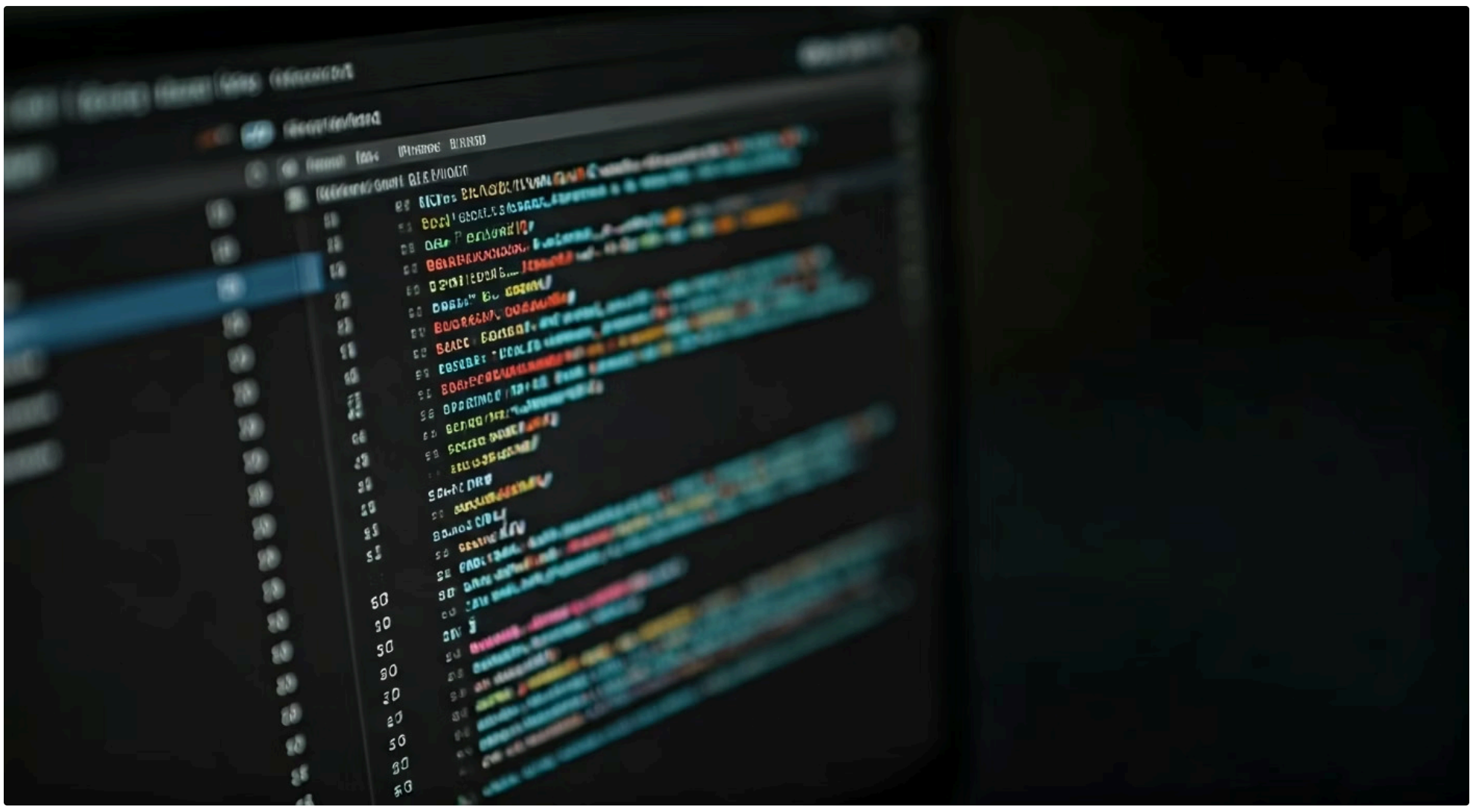
Reutiliza objetos através de pooling para economizar recursos

API Unificada

Interface consistente e familiar para todos os eventos

Essa abstração não apenas resolve problemas de compatibilidade entre navegadores, mas também otimiza o desempenho. O React reutiliza esses objetos SyntheticEvent em um processo chamado "pooling", o que significa que eles não são criados e destruídos repetidamente, economizando recursos. Ao entender e utilizar o SyntheticEvent, você ganha uma ferramenta poderosa para criar interfaces responsivas e eficientes, sem se preocupar com as peculiaridades de cada navegador.

Manipulando Eventos: Sintaxe e Boas Práticas



Compreender a natureza dos eventos em React é o primeiro passo; o próximo é saber como manipulá-los de forma eficaz. A sintaxe para adicionar ouvintes de eventos em React é muito similar ao HTML, mas com algumas diferenças cruciais. Em vez de `onclick="minhaFuncao()"`, usamos `onClick={minhaFuncao}`. Note a capitalização camelCase (`onClick` em vez de `onclick`) e o uso de chaves `{}` para passar uma referência à função, e não a chamada da função diretamente.

> ⚠️ **Atenção: Referência vs. Chamada**
>
> Passe a **referência** da função para o manipulador de eventos, garantindo que ela só seja executada quando o evento ocorrer. Use `onClick={minhaFuncao}` e não `onClick={minhaFuncao()}`.

❌ Incorreto

```
<button onClick={minhaFuncao()}>
  Clique
</button>
```

Executa a função imediatamente na renderização

✅ Correto

```
<button onClick={minhaFuncao}>
  Clique
</button>
```

Passa a referência, executa apenas no clique

Passando Argumentos

Para passar argumentos para a função, você pode usar uma arrow function, como `onClick={() => minhaFuncao(argumento)}`. Isso cria uma nova função que, quando chamada pelo evento, executa `minhaFuncao` com o argumento desejado.

01

Lógica Simples

Use arrow function inline para operações específicas de um elemento

02

Lógica Complexa

Defina funções separadas para melhor legibilidade e reutilização

03

Manutenção

Código organizado facilita futuras alterações e debugging

A escolha entre definir o manipulador de evento inline ou como uma função separada depende do contexto. Para lógica simples e específica de um elemento, uma arrow function inline pode ser conveniente. No entanto, para lógicas mais complexas ou reutilizáveis, é uma boa prática definir a função separadamente e passá-la como referência. Isso melhora a legibilidade do código e facilita a manutenção, tornando seu componente mais organizado e profissional.

Eventos Comuns e o Objeto event

Além do simples clique, o mundo da interação web é vasto, abrangendo desde o movimento do mouse até a submissão de formulários. React oferece manipuladores para uma ampla gama de eventos, como `onChange` (para inputs), `onMouseEnter`, `onMouseLeave`, `onFocus`, `onBlur`, `onSubmit`, entre muitos outros. Cada um desses eventos vem acompanhado do objeto SyntheticEvent, que é o nosso "detetive" particular, carregado de informações sobre o que acabou de acontecer.

onClick

Disparado quando o usuário clica em um elemento

onChange

Acionado quando o valor de um input é alterado

onFocus / onBlur

Quando um elemento recebe ou perde o foco

onSubmit

Disparado quando um formulário é submetido

Propriedades Importantes do Objeto event

event.target

Referência ao elemento DOM que disparou o evento

event.preventDefault()

Impede o comportamento padrão do navegador (ex: recarregar página)

event.target.value

Valor atual de inputs e outros elementos de formulário

event.stopPropagation()

Impede que o evento se propague para elementos pais

> Imagine que você está em um jogo de tabuleiro e precisa saber exatamente qual peça foi movida e para onde. O objeto event é como a ficha de registro do jogo, fornecendo todos os detalhes necessários para que você possa reagir de forma precisa.

Utilizar `event.preventDefault()` é crucial em formulários, por exemplo, para que você possa controlar a submissão dos dados via JavaScript, em vez de deixar o navegador fazer um refresh. Dominar o objeto event é fundamental para criar interações ricas e controladas em suas aplicações React.

Introdução aos Formulários em React: A Ponte de Dados

Formulários são a espinha dorsal de qualquer aplicação web interativa. Eles são a principal interface pela qual os usuários inserem dados, seja para fazer login, preencher um cadastro, enviar um comentário ou realizar uma busca. Sem formulários eficazes, a comunicação entre o usuário e a aplicação seria severamente limitada. No entanto, gerenciar o estado de formulários em aplicações JavaScript modernas, especialmente com bibliotecas como React, apresenta seus próprios desafios.

Usuário

Insere dados através da interface

Formulário

Captura e valida as informações

Estado React

Gerencia e sincroniza os dados

Backend

Processa e armazena as informações

O Desafio da Sincronização

No desenvolvimento web tradicional, o navegador lida com o estado dos elementos de formulário (o que está digitado em um input, qual opção está selecionada em um dropdown). Mas em React, onde a interface do usuário é uma função do estado, precisamos de uma maneira de sincronizar o estado interno do componente com o que o usuário vê e digita no formulário. É como ter um espelho de duas vias: o que o usuário digita é refletido no estado do React, e o que está no estado do React é refletido no formulário.

💡 Conceito-Chave

Essa sincronização é a chave para construir formulários robustos e previsíveis. Se o estado do formulário não estiver em sintonia com o estado do componente, você pode ter dados inconsistentes, validações que não funcionam ou uma experiência de usuário confusa.

A abordagem do React para resolver isso é através dos "Componentes Controlados", que veremos a seguir. Eles garantem que o estado da UI e o estado do React estejam sempre alinhados, proporcionando controle total sobre o fluxo de dados e a lógica da aplicação.

Componentes Controlados (Controlled Components): O Controle Total

No React, a forma mais comum e recomendada de lidar com formulários é através dos **Componentes Controlados**. Um componente de formulário controlado é aquele em que os dados do formulário são tratados pelo estado do componente React. Em outras palavras, o React se torna a "fonte da verdade" para os dados do formulário. Isso significa que, em vez de deixar o DOM gerenciar o estado do input, o React assume essa responsabilidade, mantendo o valor do input no estado do componente.

> Imagine um boneco de marionete. Cada movimento do boneco é controlado por um mestre que puxa as cordas. No contexto de um componente controlado, o input do formulário é o boneco, e o estado do React é o mestre que puxa as cordas.

Como Funciona?

Usuário Digita

Entrada de dados no input

onChange Dispara

Evento detecta a mudança

UI Re-renderiza

Input reflete o novo valor

Estado Atualiza

React atualiza o estado interno

Requisitos para Implementação

1

Propriedade value

O atributo `value` do elemento de formulário deve ser definido para o valor do estado do React

2

Manipulador onChange

Um manipulador de evento `onChange` deve ser anexado ao elemento para atualizar o estado sempre que o valor mudar

Essa abordagem garante que o estado da aplicação e o que é exibido na interface estejam sempre sincronizados, permitindo validações em tempo real, desabilitação condicional de botões e manipulação programática do formulário com facilidade.

Implementando Componentes Controlados na Prática

Vamos solidificar o conceito de componentes controlados com um exemplo prático. Para criar um input de texto controlado, precisamos de um estado para armazenar seu valor e uma função para atualizá-lo. Usaremos o hook `useState` para gerenciar esse estado em um componente funcional.

Passo a Passo

Definir o Estado

Criar um estado inicial para o input usando `useState`

Vincular o value

Conectar a propriedade `value` do input ao estado

Adicionar onChange

Anexar manipulador que atualiza o estado a cada mudança

Exemplo Completo

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

function MeuFormulario() {
  const [nome, setNome] = useState('');

  const handleChange = (event) => {
    setNome(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Nome submetido:', nome);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="nomeInput">Nome:</label>
      <input
        type="text"
        id="nomeInput"
        value={nome}
        onChange={handleChange}
      />
      <p>Você digitou: {nome}</p>
      <button type="submit">Enviar</button>
    </form>
  );
}

export default MeuFormulario;
```

🔄 Ciclo de Feedback Contínuo

Neste exemplo, cada vez que o usuário digita algo no campo "Nome", a função `handleChange` é acionada. Ela pega o novo valor do input (`event.target.value`) e o usa para atualizar o estado `nome` através de `setNome`. Como o `value` do input está vinculado a `nome`, o input é redesenhado com o novo valor, completando o ciclo de controle.

Essa abordagem de feedback contínuo garante que o estado do React esteja sempre em sincronia com o que o usuário vê e interage na tela.

Vantagens dos Componentes Controlados

A adoção de componentes controlados pode parecer um pouco mais trabalhosa inicialmente do que deixar o DOM gerenciar o estado, mas os benefícios a longo prazo são imensos. Eles oferecem um controle sem precedentes sobre o comportamento do formulário, transformando-o em uma parte previsível e poderosa da sua aplicação. Essa previsibilidade é o que permite construir interfaces de usuário mais ricas e robustas.

Validação em Tempo Real

Feedback instantâneo enquanto o usuário digita, sem esperar pela submissão

Manipulação Programática

Preencher campos automaticamente, aplicar máscaras, formatar dados dinamicamente

Estado Consistente

UI e estado sempre sincronizados, eliminando inconsistências

Controle Condicional

Desabilitar botões, mostrar/ocultar campos baseado em condições

Exemplos Práticos

Validação Instantânea

Mostrar mensagem de erro assim que o usuário digita um e-mail inválido, sem precisar esperar pela submissão do formulário.

Formatação Automática

Adicionar máscaras para CPF, telefone ou CEP enquanto o usuário digita, melhorando a experiência.

> Imagine um carro com um sistema de diagnóstico em tempo real. Cada sensor envia dados para um computador central que monitora e ajusta o desempenho. Componentes controlados funcionam de forma similar: o estado do React é o computador central, e cada input é um sensor.

Essa capacidade de intervir e moldar a experiência do usuário em tempo real é uma das maiores vantagens, levando a formulários mais inteligentes, acessíveis e com uma usabilidade superior.

Gerenciando Múltiplos Inputs com Componentes Controlados

Em aplicações reais, formulários raramente contêm apenas um campo. Formulários de cadastro, login ou perfil de usuário podem ter dezenas de inputs. Gerenciar o estado de cada um deles individualmente com `useState` para cada campo (nome, email, senha) pode rapidamente levar a um código repetitivo e difícil de manter. Felizmente, React oferece uma maneira elegante de lidar com múltiplos inputs usando um único objeto de estado.

> Pense em uma ficha de cadastro onde cada campo tem um rótulo único. Em vez de ter uma gaveta separada para cada informação (nome, email, telefone), você pode ter uma única pasta "Dados do Usuário" e dentro dela, fichas etiquetadas para cada detalhe.

Estratégia de Implementação

01

Estado como Objeto

Inicializar o estado com um objeto contendo todas as propriedades do formulário

02

Atributo name

Adicionar atributo `name` em cada input correspondente à chave do objeto

03

Manipulador Genérico

Criar um único `onChange` que atualiza a propriedade correta usando `event.target.name`

Código de Exemplo

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

function FormularioMultiplo() {
  const [formData, setFormData] = useState({
    nome: '',
    email: '',
    senha: ''
  });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormData(prevFormData => ({
      ...prevFormData,
      [name]: value
    }));
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Dados do formulário:', formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="nome"
        value={formData.nome}
        onChange={handleChange}
      />
      <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
      />
      <input
        type="password"
        name="senha"
        value={formData.senha}
        onChange={handleChange}
      />
      <button type="submit">Cadastrar</button>
    </form>
  );
}
```

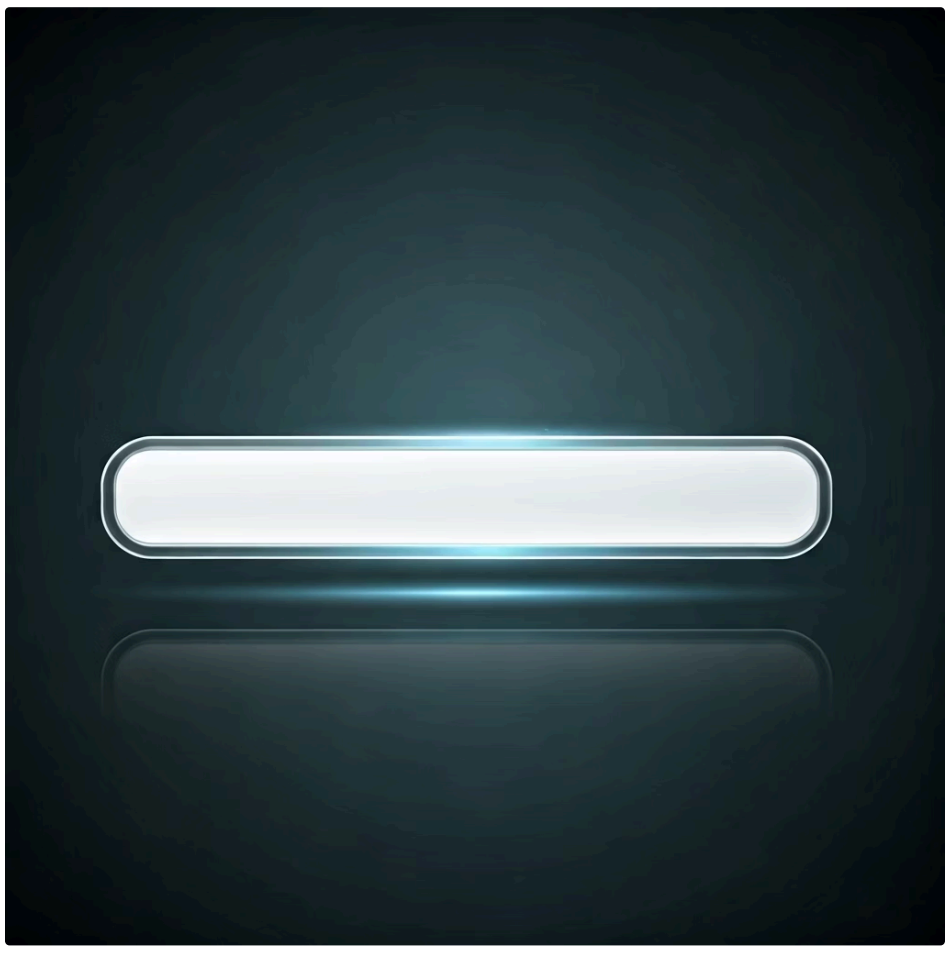
✨ Vantagem da Abordagem

Usamos a sintaxe de spread (`...formData`) para copiar o estado anterior e sobrescrever apenas a propriedade correspondente ao input que mudou. Isso mantém o código limpo e escalável, mesmo para formulários complexos.

Componentes Controlados para textarea e select

A beleza dos componentes controlados em React é que a mesma lógica fundamental se aplica a diferentes tipos de elementos de formulário, embora com pequenas nuances. Além dos inputs de texto, é comum precisarmos controlar `textarea` (para textos longos) e `select` (para listas de opções). A boa notícia é que o princípio de vincular o `value` ao estado e usar `onChange` para atualizá-lo permanece o mesmo.

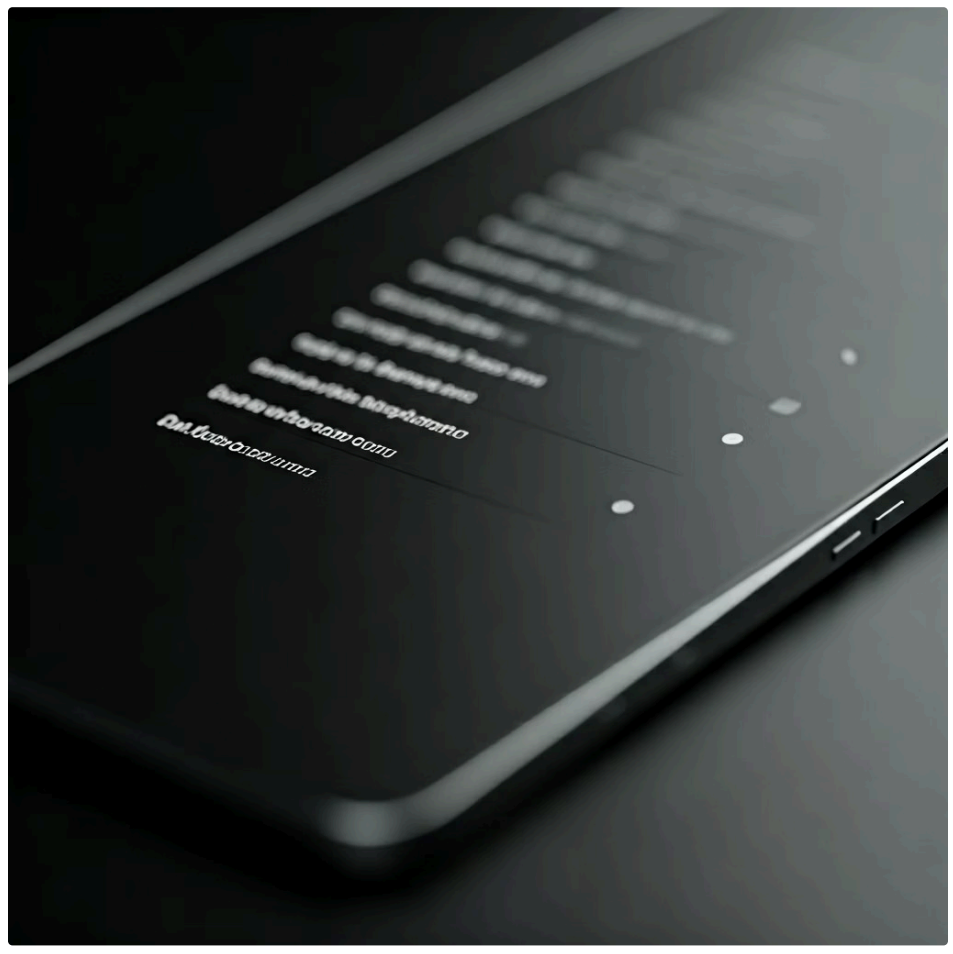
textarea



Para um `textarea`, a implementação é idêntica a um input de texto. Em vez de colocar o texto entre as tags `<textarea>...</textarea>`, você define o valor usando a propriedade `value`.

```
<textarea
  name="mensagem"
  value={formData.mensagem}
  onChange={handleChange}
/>
```

select



Para o elemento `select`, você define a propriedade `value` no próprio elemento `<select>`. O React se encarrega de selecionar a opção cujo `value` corresponde ao estado.

```
<select
  name="cidade"
  value={formData.cidade}
  onChange={handleChange}
>
  <option value="">Selecione</option>
  <option value="sp">São Paulo</option>
  <option value="rj">Rio de Janeiro</option>
</select>
```

textarea

- Use `value` em vez de conteúdo entre tags
- Ideal para textos longos e comentários
- Suporta quebras de linha automaticamente

select

- Defina `value` no elemento `<select>`
- React seleciona a opção correspondente
- Para múltiplas seleções, use array no `value`

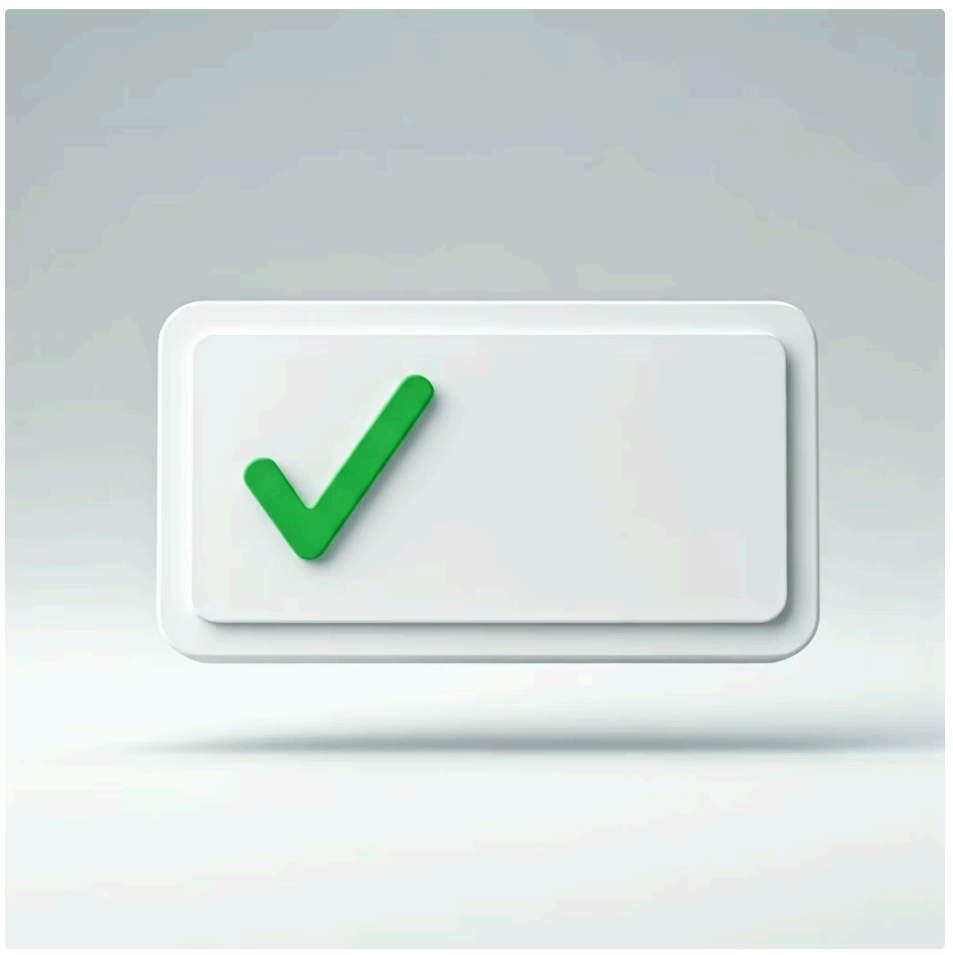
> Pense nisso como diferentes ferramentas em uma caixa de ferramentas: cada uma tem sua forma específica de uso, mas todas servem ao mesmo propósito de construir e controlar.

Ao aplicar o padrão de componente controlado a `textarea` e `select`, você mantém a consistência em seu código e garante que todos os elementos do seu formulário estejam sob o controle do estado do React, facilitando a validação e o processamento dos dados.

Lidando com Checkboxes e Radio Buttons

Checkboxes e radio buttons introduzem uma pequena variação na forma como gerenciamos seus estados, pois eles lidam com valores booleanos ou seleções mutuamente exclusivas. No entanto, o conceito de componente controlado ainda se aplica, apenas com uma propriedade diferente para o valor e uma forma específica de acessar a mudança.

Checkbox



Para um **checkbox**, em vez de `value`, usamos a propriedade `checked` para determinar se ele está marcado ou não. O valor do estado associado a um checkbox geralmente é um booleano (`true` ou `false`).

```
const [aceitoTermos, setAceitoTermos] =
useState(false);

const handleCheckboxChange = (event) => {
  setAceitoTermos(event.target.checked);
};

<input
  type="checkbox"
  name="termos"
  checked={aceitoTermos}
  onChange={handleCheckboxChange}
/>
<label>Aceito os termos</label>
```

Radio Button

Radio buttons funcionam em grupos, onde apenas um pode ser selecionado por vez. Para controlá-los, você define a propriedade `checked` para o radio button que corresponde ao valor atual do seu estado.

```
const [genero, setGenero] = useState('');

const handleRadioChange = (event) => {
  setGenero(event.target.value);
};

<input
  type="radio"
  name="genero"
  value="masculino"
  checked={genero === 'masculino'}
  onChange={handleRadioChange}
/>
<label>Masculino</label>

<input
  type="radio"
  name="genero"
  value="feminino"
  checked={genero === 'feminino'}
  onChange={handleRadioChange}
/>
<label>Feminino</label>
```

Checkbox	Radio Button
Use `checked` em vez de `value`	Use `checked` baseado em comparação
Acesse `event.target.checked` para o estado	Acesse `event.target.value` para o valor selecionado
Valor geralmente é booleano	Apenas um pode estar ativo por grupo

> Pense nos checkboxes como interruptores de luz individuais (ligar/desligar) e nos radio buttons como as estações de rádio de um aparelho antigo (você só pode ouvir uma por vez).

Embora a mecânica de `checked` e `event.target.checked` seja específica, a filosofia de controle do estado pelo React permanece consistente, garantindo que você tenha total domínio sobre as escolhas do usuário.

Submissão de Formulários e Prevenção de Comportamento Padrão

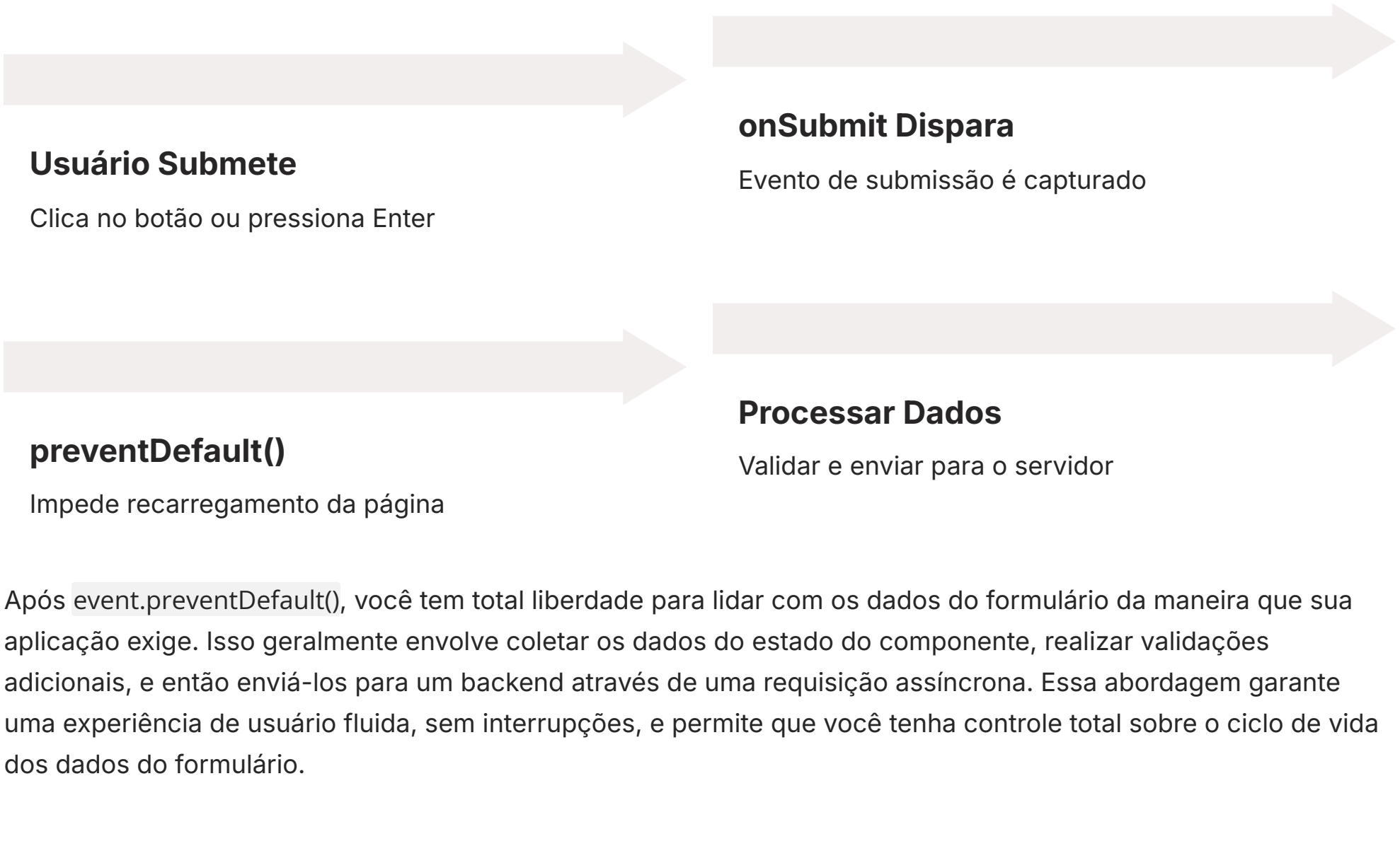
Depois de coletar todos os dados através dos inputs controlados, o próximo passo crucial é processar a submissão do formulário. Em HTML puro, quando um formulário é submetido (por exemplo, clicando em um botão `<button type="submit">` ou pressionando Enter em um campo de texto), o navegador tenta recarregar a página e enviar os dados para o servidor. Em aplicações React, que são geralmente Single Page Applications (SPAs), esse comportamento padrão não é desejado, pois causaria um refresh completo e a perda do estado da aplicação.

🛑 Comportamento Padrão do Navegador

Para evitar esse recarregamento indesejado, usamos o evento `onSubmit` no elemento `<form>` e, dentro do manipulador, chamamos `event.preventDefault()`. Essa função instrui o navegador a não executar sua ação padrão de submissão.

Exemplo Completo de Submissão

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

function FormularioSubmissao() {
  const [email, setEmail] = useState('');

  const handleEmailChange = (event) => {
    setEmail(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault(); // CRÍTICO!
    console.log('Email submetido:', email);

    // Aqui você faria uma requisição HTTP
    // (e.g., com Fetch ou Axios)
    // para enviar 'email' para um servidor.

    setEmail(''); // Limpar o campo após submissão
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="emailInput">Email:</label>
      <input
        type="email"
        id="emailInput"
        value={email}
        onChange={handleEmailChange}
      />
      <button type="submit">Assinar Newsletter</button>
    </form>
  );
}
```



Após `event.preventDefault()`, você tem total liberdade para lidar com os dados do formulário da maneira que sua aplicação exige. Isso geralmente envolve coletar os dados do estado do componente, realizar validações adicionais, e então enviá-los para um backend através de uma requisição assíncrona. Essa abordagem garante uma experiência de usuário fluida, sem interrupções, e permite que você tenha controle total sobre o ciclo de vida dos dados do formulário.

Validação de Formulários e Acessibilidade (A11Y)

Construir formulários não é apenas sobre coletar dados; é sobre coletá-los de forma correta e garantir que todos os usuários, independentemente de suas habilidades ou tecnologias assistivas, possam interagir com eles. A validação de formulários e a acessibilidade (A11Y) são dois pilares fundamentais para criar experiências de usuário de alta qualidade.

Validação

A **validação** garante que os dados inseridos pelo usuário estejam no formato esperado e sejam válidos antes de serem processados. Em React, com componentes controlados, a validação pode ser feita em tempo real, enquanto o usuário digita.

- Feedback imediato ao usuário
- Validação durante `onChange` ou `onBlur`
- Mensagens de erro armazenadas no estado

Acessibilidade (A11Y)

A **acessibilidade** garante que seus formulários sejam utilizáveis por pessoas com deficiência, incluindo usuários de leitores de tela e outras tecnologias assistivas.

- Tags semânticas (`<label>`, `htmlFor`)
- Atributos ARIA (`aria-describedby`, `aria-invalid`)
- Feedback visual claro para erros

Exemplo de Validação e Acessibilidade

```
const [email, setEmail] = useState('');
const [erroEmail, setErroEmail] = useState('');

const validarEmail = (email) => {
  if (!email.includes('@')) {
    setErroEmail('Email inválido. Deve conter "@".');
  } else {
    setErroEmail('');
  }
};

const handleEmailChange = (event) => {
  const novoEmail = event.target.value;
  setEmail(novoEmail);
  validarEmail(novoEmail);
};

<label htmlFor="emailInput">Email:</label>
<input
  type="email"
  id="emailInput"
  value={email}
  onChange={handleEmailChange}
  aria-invalid={erroEmail ? "true" : "false"}
  aria-describedby="emailErro"
/>
{erroEmail && (
  <p id="emailErro" style={{ color: 'red' }}>
    {erroEmail}
  </p>
)}
```

Validação em Tempo Real

Feedback instantâneo enquanto o usuário digita

Atributos ARIA

Comunicação clara com tecnologias assistivas

Feedback Visual

Mensagens de erro claras e destacadas

Conformidade

Atende padrões de qualidade e inclusão

> Integrar validação e acessibilidade desde o início não é apenas uma boa prática, mas uma responsabilidade. Isso não só melhora a experiência para todos os usuários, mas também contribui para a qualidade geral e a conformidade da sua aplicação.

Otimização e Tendências: Vite e Performance Web

No cenário atual do desenvolvimento frontend, a velocidade e a eficiência são cruciais, tanto para o desenvolvedor quanto para o usuário final. Ferramentas como o Vite surgiram como uma resposta a essa demanda, oferecendo uma experiência de desenvolvimento significativamente mais rápida em comparação com configuradores mais antigos e complexos como o Webpack. Ao construir formulários e manipular eventos, a escolha da sua ferramenta de build pode impactar diretamente sua produtividade e a performance da aplicação.

Vite: Velocidade no Desenvolvimento

Módulos ES Nativos

Elimina empacotamento pesado durante desenvolvimento

HMR Instantâneo

Hot Module Replacement quase instantâneo

Build Otimizado

Produção otimizada com Rollup

Core Web Vitals

LCP

Largest Contentful Paint

Velocidade de carregamento do conteúdo principal

FID

First Input Delay

Tempo de resposta à primeira interação

CLS

Cumulative Layout Shift

Estabilidade visual durante carregamento

> É como ter um carro esportivo para o seu desenvolvimento, que acelera de 0 a 100 em segundos.

Impacto nos Formulários

Carregamento Rápido

Formulários aparecem rapidamente (LCP)

Resposta Instantânea

Interações sem delay (FID)

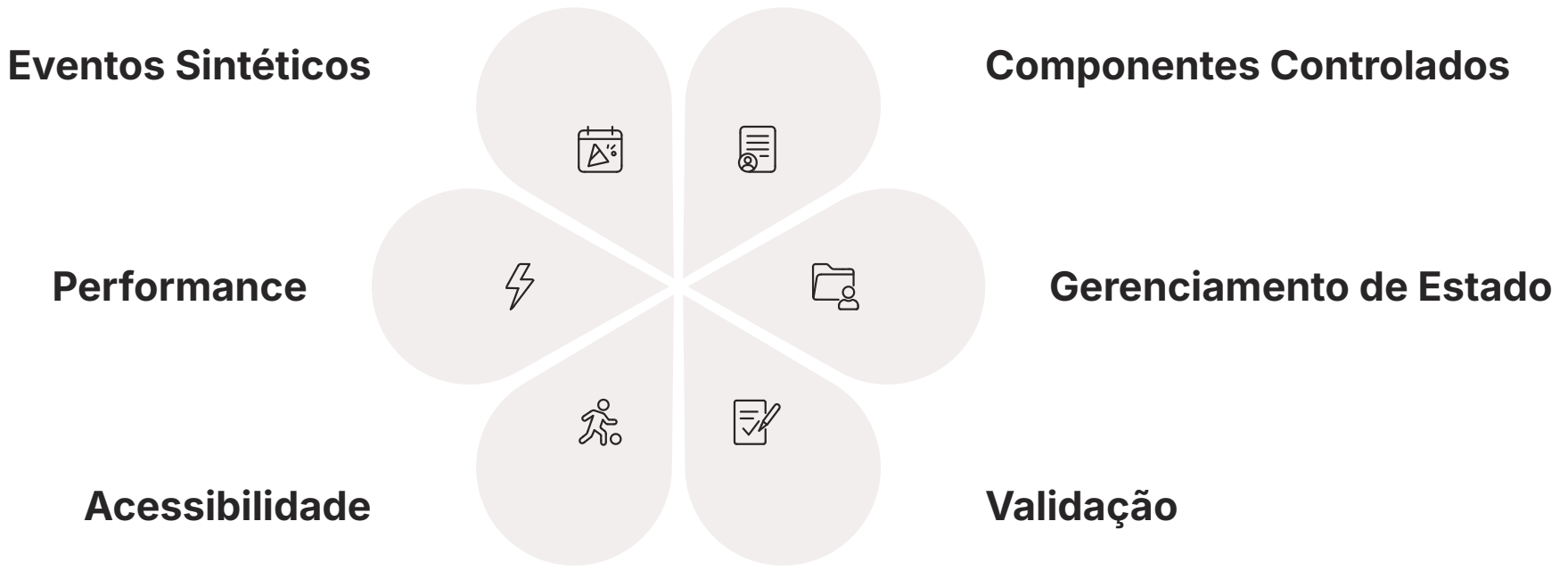
Layout Estável

Sem mudanças inesperadas (CLS)

Formulários bem construídos e otimizados contribuem diretamente para essas métricas. Por exemplo, um formulário que carrega rapidamente (LCP), responde instantaneamente às interações do usuário (FID) e não causa mudanças inesperadas de layout (CLS) oferece uma experiência superior. Ferramentas modernas e práticas de código limpo, como as que usamos para componentes controlados, são essenciais para atingir esses objetivos.

Consolidação e Próximos Passos

Chegamos ao fim de uma aula fundamental para a construção de interfaces interativas em React. Percorremos o caminho desde a compreensão dos eventos sintéticos, que atuam como um tradutor universal para as interações do usuário, até a maestria na criação de formulários controlados, onde o React assume o controle total do estado dos inputs. Vimos como gerenciar múltiplos campos de forma eficiente, lidar com diferentes tipos de inputs como textarea, select, checkboxes e radio buttons, e a importância de submeter formulários de maneira controlada, prevenindo o comportamento padrão do navegador.



Mais do que apenas funcionalidade, enfatizamos a importância da validação em tempo real e da acessibilidade (A11Y), garantindo que suas aplicações sejam robustas e inclusivas para todos os usuários. Por fim, conectamos esses conceitos com as tendências modernas de desenvolvimento, como o uso do Vite para otimizar o fluxo de trabalho e a atenção à Performance Web para entregar experiências rápidas e fluidas.

Em Prática: Checklist Essencial

event.preventDefault()

Sempre use em `onSubmit` para evitar recarregamento da página

Componentes Controlados

Prefira para ter total domínio sobre o estado do formulário

Estado Unificado

Utilize um único objeto de estado para gerenciar múltiplos inputs

Acessibilidade

Não se esqueça das labels e atributos ARIA

Testes

Teste seus formulários com diferentes entradas e cenários

Autoavaliação

1

Qual é a principal função do SyntheticEvent em React?

- a) Substituir completamente os eventos nativos do navegador por uma API React.
- b) Padronizar o comportamento dos eventos entre diferentes navegadores.
- c) Aumentar a segurança dos eventos, prevenindo ataques XSS.
- d) Otimizar a renderização de componentes que dependem de eventos.

2

Para que serve o método event.preventDefault() em um manipulador de evento onSubmit de um formulário React?

- a) Ele impede que o formulário seja validado pelo navegador.
- b) Ele impede que o evento se propague para elementos pais.
- c) Ele impede o comportamento padrão do navegador, como o recarregamento da página.
- d) Ele limpa todos os campos do formulário após a submissão.

3

Qual das seguintes opções descreve corretamente um "Componente Controlado" em React?

- a) Um componente que não permite a interação do usuário.
- b) Um componente de formulário cujo valor é gerenciado pelo estado do React.
- c) Um componente que só pode ser atualizado por um componente pai.
- d) Um componente que utiliza exclusivamente o DOM para gerenciar seu estado.

4

Ao lidar com múltiplos inputs em um formulário controlado, qual é a abordagem mais recomendada para gerenciar o estado de forma eficiente?

- a) Criar um useState separado para cada input individualmente.
- b) Usar um único objeto no estado do React, mapeando name dos inputs às suas propriedades.
- c) Deixar o DOM gerenciar o estado dos inputs e acessá-los apenas na submissão.
- d) Utilizar useRef para cada input e acessar seus valores diretamente.

5

Questão Dissertativa

Explique a importância da acessibilidade (A11Y) na construção de formulários web em React e cite um exemplo prático de como implementá-la.

Gabarito

Questão 1

Resposta: b) Padronizar o comportamento dos eventos entre diferentes navegadores.

Questão 2

Resposta: c) Ele impede o comportamento padrão do navegador, como o recarregamento da página.

Questão 3

Resposta: b) Um componente de formulário cujo valor é gerenciado pelo estado do React.

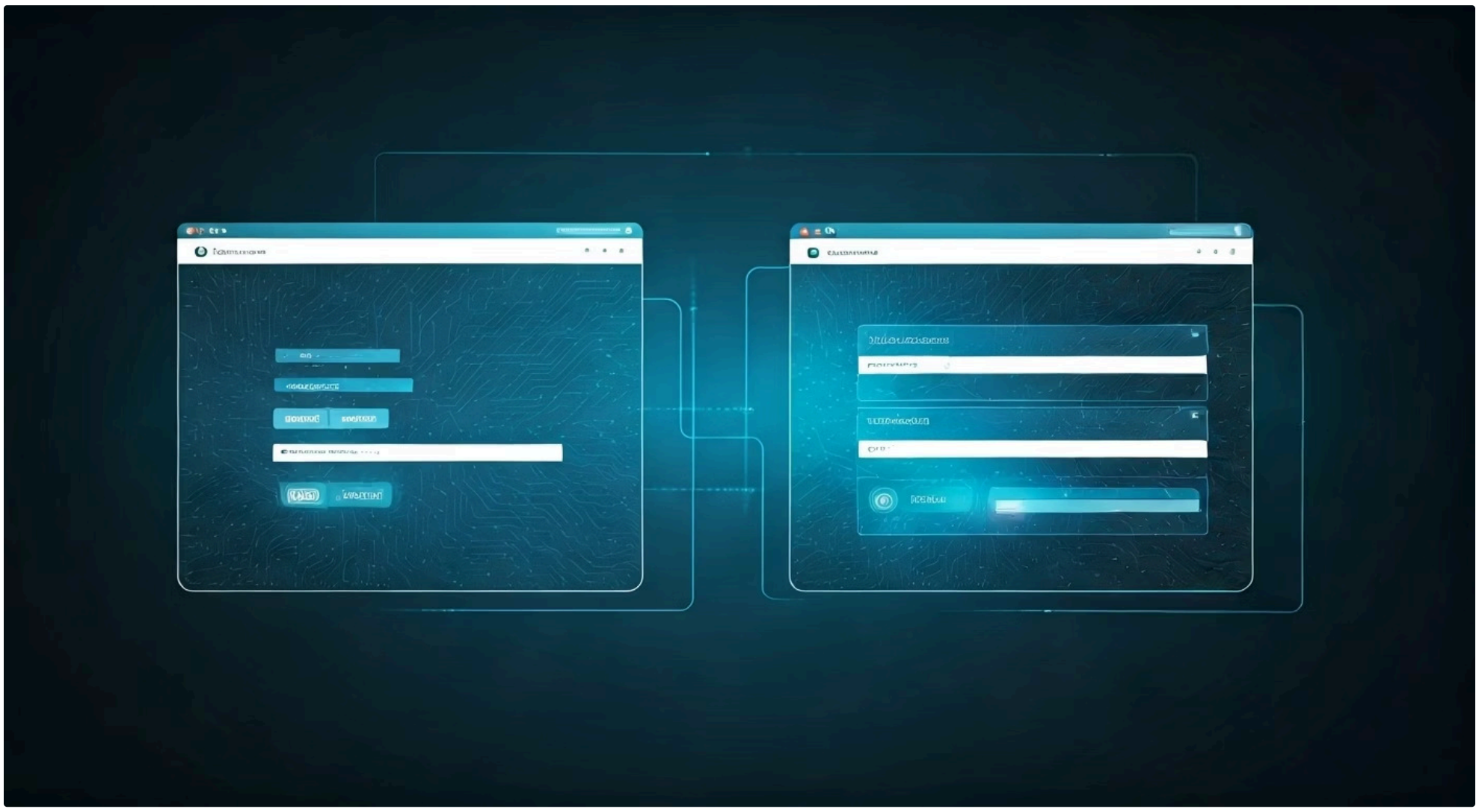
Questão 4

Resposta: b) Usar um único objeto no estado do React, mapeando name dos inputs às suas propriedades.

📝 Nota sobre a Questão Dissertativa

A questão 5 é dissertativa e deve abordar a importância da acessibilidade para garantir que todos os usuários, incluindo aqueles com deficiências, possam interagir com formulários. Um exemplo prático seria o uso de atributos `aria-describedby` e `aria-invalid` para associar mensagens de erro a campos de input, permitindo que leitores de tela comuniquem claramente os problemas aos usuários.

Conexão com a Próxima Aula



Agora que você dominou a arte de coletar dados e interagir com o usuário através de eventos e formulários, o próximo passo natural é aprender a enviar e receber dados de serviços externos. Na **Aula 25 – Consumindo APIs com Fetch**, exploraremos como sua aplicação React pode se comunicar com o mundo exterior, buscando informações e enviando os dados coletados em seus formulários para um backend, tornando suas aplicações verdadeiramente dinâmicas e conectadas.

Aula 24

Eventos e Formulários

Aula 25

Consumindo APIs com Fetch

Resultado

Aplicações Conectadas

Recursos Adicionais

📚 Documentação Oficial do React sobre Eventos

Para aprofundar nos detalhes do SyntheticEvent e outros manipuladores.

📝 Documentação Oficial do React sobre Formulários

Para explorar mais a fundo os componentes controlados e não controlados.

♿ MDN Web Docs sobre Acessibilidade em Formulários

Para diretrizes detalhadas sobre como tornar seus formulários inclusivos.

⚡ Documentação do Vite

Para entender melhor como essa ferramenta otimiza o desenvolvimento frontend.

⚖️ NOTA IMPORTANTE

As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.