



React

Frontend

Formulários em react

O evento onSubmit em React pode ser um obstáculo para a manipulação das informações de entrada de um formulário, pois força o carregamento da página, o que pode resultar na perda das informações de estado.

Para contornar esse problema, é necessário impedir o comportamento padrão de envio do formulário. Isso pode ser feito chamando o método preventDefault() no objeto de evento event dentro do manipulador de eventos onSubmit.

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

export default function MyForm() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault(); // Previne o envio padrão

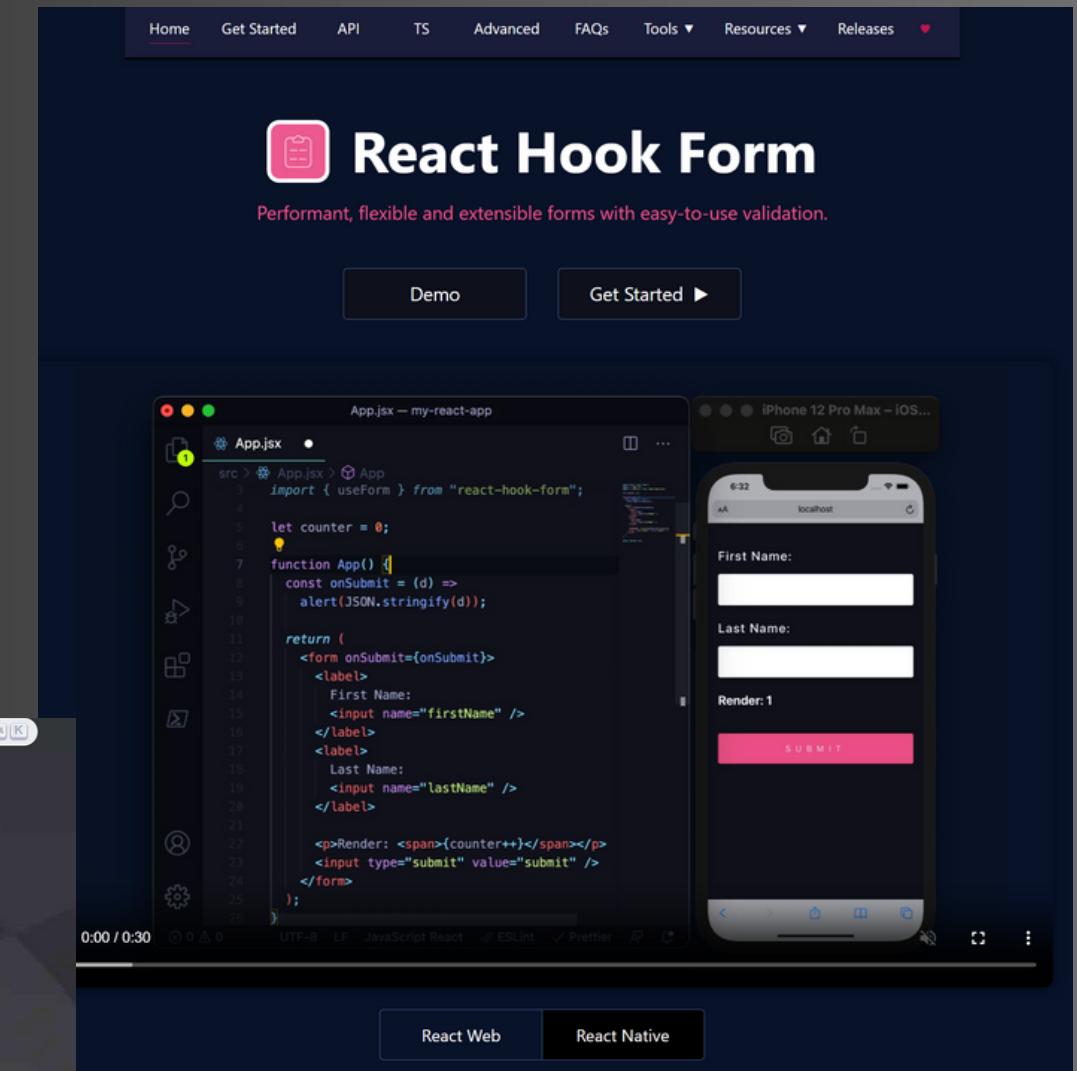
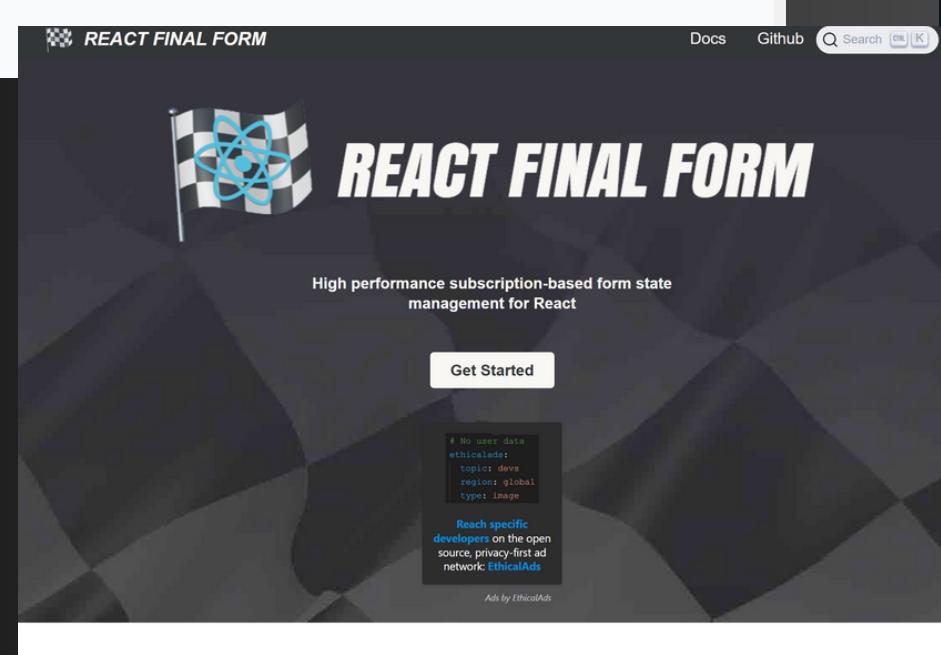
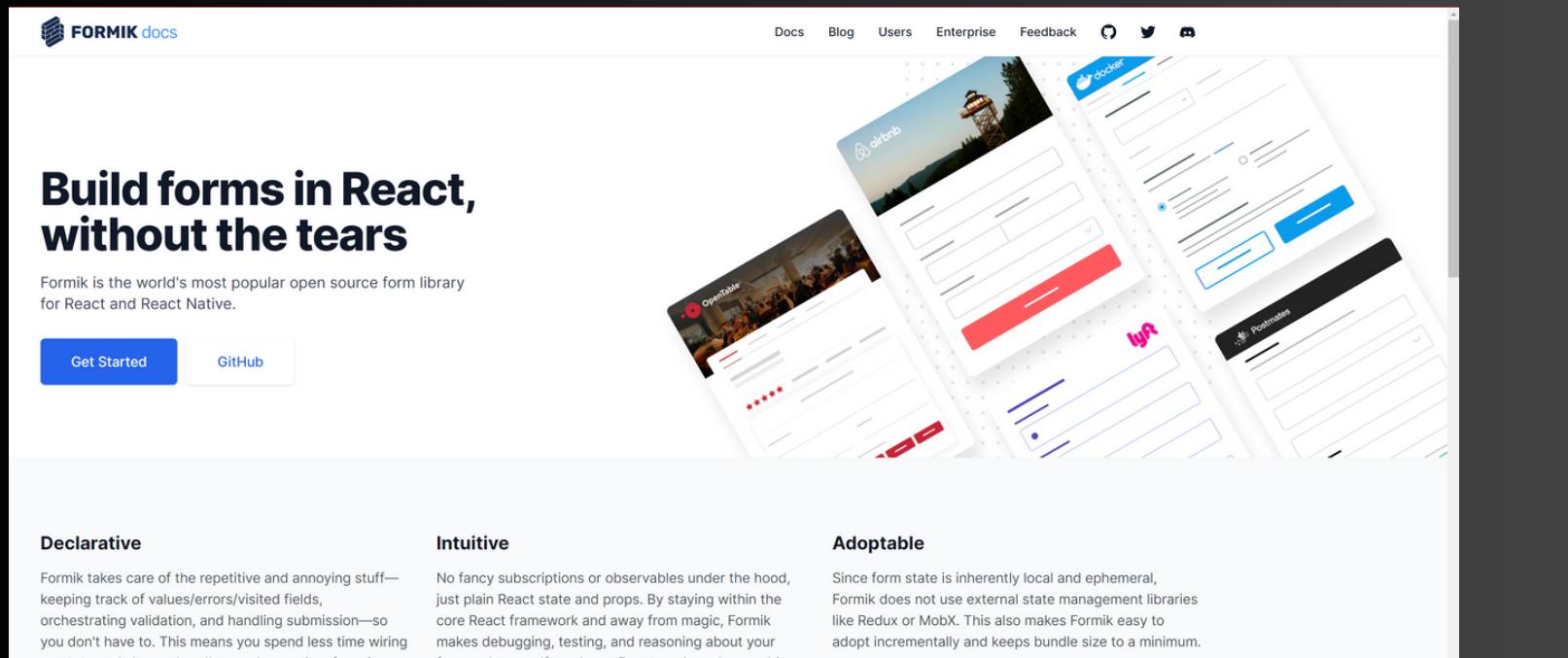
    console.log("Nome:", name); // Faça algo com os dados do formulário
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      <button type="submit">Enviar</button>
    </form>
  );
}
```

Link para codesandbox:
<https://codesandbox.io/s/moosh-leaf-hglgp7?file=/src/MyForm/index.jsx>

Formulários em react

Outra maneira de contornar o problema é utilizar bibliotecas voltadas para manipulação de formulários como : React Hook Form, Formik e Final Form



useState com objetos

Para manipular os estados de campos de input, é recomendável evitar a criação de um estado separado para cada campo. Em vez disso, é possível criar um objeto com as chaves sendo os nomes dos campos.

Ao invés de utilizar:

```
export default function App() {
  const [nome, setNome] = useState("");
  const [email, setEmail] = useState("");
  const [telefone, setTelefone] = useState("");
  return (
    <div className="App">
      <form>
        <label htmlFor="nome">
          Nome:
          <input name="nome" />
        </label>
        <label htmlFor="email">
          Email:
          <input name="email" />
        </label>
        <label htmlFor="telefone">
          Telefone:
          <input name="telefone" />
        </label>
        <button>Criar usuário</button>
      </form>
    </div>
  );
}
```

Utilize:

```
import { useState } from "react";
import "./styles.css";

export default function App() {
  const [user, setUser] = useState({ nome: "", email: "", telefone: "" });
  return (
    <div className="App">
      <form>
        <label htmlFor="nome">
          Nome:
          <input name="nome" />
        </label>
        <label htmlFor="email">
          Email:
          <input name="email" />
        </label>
        <label htmlFor="telefone">
          Telefone:
          <input name="telefone" />
        </label>
        <button>Criar usuário</button>
      </form>
    </div>
  );
}
```

useState com objeto

Para manipular o conteúdo do objeto, pode-se utilizar a callback presente no setState para obter estado anterior(preview) e a partir desestruturar o objeto e então sobrescrever as informações conforme o campo do input é modificado.

```
const [user, setUser] = useState({ nome: "", email: "", telefone: "" });
function handleForm(e) {
  setUser((preview) => ({ ...preview, chave: "valor do campo modificado" }));
}
return (
  <div className="App">
    <form>
      <label htmlFor="nome">
        Nome:
        <input
          name="nome"
          onChange={(e) => {
            handleForm;
          }}
        />
      </label>
    </form>
  </div>
)
```

useState Objeto

Para manipular de forma dinâmica os campos de input utilize os atributos “name” e “value” do evento do input.

```
function handleForm(e) {  
    //nome do input      valor do input  
    setUser((preview) => ({ ...preview, [e.target.name]: e.target.value }));  
}
```

e.target.name (Traz o nome do input ex: nome, email, telefone)

e.target.email (Traz o valor do input ex: valor digitado de nome, email, ou telefone)

Vamos à prática

Persistência Local

LocalStorage

Cria um objeto que fica armazenado na memória local que não expira. Para apagá-lo é necessário apagar o cache do navegador.

O Local Storage é limitado a 5 MB de dados.

SessionStorage

Cria um objeto que fica armazenado na memória local que expira ao encerrar a sessão.
“O Session Storage não tem limite de tamanho.” - Nesse caso, não tem limitação padrão.

Cookies

Pequenas informações em peças que o servidor envia ao web browser para conferir posteriormente se as requisições partem do mesmo browser. Ideais para manuseio de sessões, personalização e rastreio.

Os Cookies são limitados a 4096 bytes de dados.

Persistência Local

Local Storage, Cookies e Session podem ser usados para uma variedade de propósitos, incluindo:

- Armazenar preferências do usuário
- Manter o estado de login do usuário
- Armazenar carrinhos de compras
- Armazenar informações de rastreamento

A escolha do método correto para armazenar dados depende das necessidades específicas da aplicação.

Como salvar o objeto localmente com LocalStorage

```
localStorage.setItem(key, value);
```

Como obter objeto salvo localmente

```
let lastname = localStorage.getItem(key);
```

Como remover objeto salvo localmente

```
localStorage.removeItem(key);
```

Como remover todos os objetos localmente

```
localStorage.clear();
```

Como salvar o objeto localmente com SessionStorage

```
sessionStorage.setItem("key", "value");
```

Como obter objeto salvo localmente

```
let data = sessionStorage.getItem("key");
```

Como remover objeto salvo localmente

```
sessionStorage.removeItem("key");
```

Como remover todos os objetos localmente

```
sessionStorage.clear();
```

Associando useState e localStorage para criar um estado com persistência

Um problema comum na manipulação de estado é a persistência de informações ao recarregar a página. Isso pode ser um problema para informações como o tema da aplicação, informações do usuário e informações do carrinho de compras.

A seguir temos um exemplo de script para manipulação de estado para renderização de tema:

```
import { useState } from "react";
import "./styles.css";

export default function App() {
  const [theme, setTheme] = useState("claro");

  function handleTheme() {
    setTheme((prevTheme) => (prevTheme === "claro" ? "escuro" : "claro"));
  }

  return (
    <div
      className="App"
      style={{ backgroundColor: theme === "claro" ? "white" : "black" }}
    >
      <button onClick={handleTheme}>
        {theme === "claro" ? "escuro" : "claro"}
      </button>
    </div>
  );
}
```

Ao recarregar essa página o tema é resetado para o estado “claro”

Para contornar isso o tema é obtido a partir do local storage

Obtendo tema a partir do local storage

O tema obtido na construção do componente através do localStorage

```
const [theme, setTheme] = useState(localStorage.getItem("theme") || "claro");
```

Ao mudar o estado, essa informação é salva no localStorage na atualização do componente

```
useEffect(() => {
  localStorage.setItem("theme", theme);
}, [theme]);
```

Vamos à prática