

Utilizando Session Storage em React para Gerenciar Estados Temporários

Por Escola Dnc

Introdução

Neste ebook, exploraremos o uso do Session Storage em aplicações React para gerenciar estados temporários de forma eficiente. Abordaremos conceitos importantes como a diferença entre Local Storage e Session Storage, implementação prática em um componente de lista de projetos, e boas práticas para persistência de dados entre rotas. O foco será em criar uma experiência de usuário interativa mantendo dados relevantes apenas durante a sessão atual.

Características principais

- ## Quando usar cada um

- Session Storage é ideal para armazenar estados temporários como curtidas em uma lista de projetos durante uma única sessão.

Implementando Session Storage em um Componente React

Estrutura básica do componente

```
import React, { useState, useEffect } from 'react';import { Button
} from './components';const ProjectList = () => {  const
[faveProjects, setFaveProjects] = useState([]);    // Restante do
código...}
```

Função para gerenciar projetos favoritos

```
const handleSavedProjects = (id) => {
  setFaveProjects((prevFaveProjects) => {    if
  (prevFaveProjects.includes(id)) {      const filteredArray =
  prevFaveProjects.filter(projectId => projectId !== id);
  sessionStorage.setItem('faveProjects',
  JSON.stringify(filteredArray));      return filteredArray;    }
  else {      const newFaveProjects = [...prevFaveProjects, id];
  sessionStorage.setItem('faveProjects',
  JSON.stringify(newFaveProjects));      return newFaveProjects;    }
  });};
```

Carregando dados do Session Storage

```
useEffect(() => {  const savedProjects =
  JSON.parse(sessionStorage.getItem('faveProjects'));  if
  (savedProjects) {    setFaveProjects(savedProjects);  }}, []);
```

Renderização condicional dos ícones de curtida

```
{projects.map(project => ( <Button onClick={() =>
handleSavedProjects(project.id)} className="unstyle"> <img
src={faveProjects.includes(project.id) ? likeFilled : likeOutline}
alt="Like" /> </Button>)))}
```

Vantagens do Session Storage neste Contexto

1. **Persistência entre rotas:** Os dados permanecem disponíveis mesmo ao navegar entre diferentes páginas da aplicação.
2. **Limpeza automática:** Ao fechar a aba ou o navegador, os dados são automaticamente removidos, evitando acúmulo desnecessário.
3. **Experiência do usuário:** Permite manter o estado de interações (como curtidas) durante toda a sessão, sem necessidade de login.
4. **Performance:** Acesso rápido aos dados, sem necessidade de requisições ao servidor para recuperar o estado.
5. **Simplicidade:** Fácil de implementar e gerenciar, sem necessidade de configurações complexas.

O uso do Session Storage neste caso proporciona uma experiência fluida ao usuário, mantendo suas interações consistentes durante toda a navegação no site.

Considerações e Boas Práticas

Limitações do Session Storage

- **Capacidade de armazenamento:** Geralmente limitada a 5-10MB, dependendo do navegador.
- **Segurança:** Dados não são criptografados, evite armazenar informações sensíveis.

Dicas de implementação

1. **Validação de dados:** Sempre verifique se os dados existem no Session Storage antes de usá-los.
2. **Tratamento de erros:** Implemente try-catch ao trabalhar com `JSON.parse` para evitar quebras por dados inválidos.
3. **Limpeza de dados:** Considere implementar uma função para limpar dados obsoletos do Session Storage.
4. **Versionamento:** Se a estrutura dos dados mudar, considere incluir uma versão para facilitar migrações futuras.
5. **Fallback:** Tenha uma estratégia alternativa caso o Session Storage não esteja disponível no navegador do usuário.

Exemplo de implementação robusta

```
const getSavedProjects = () => { try { const savedProjects =
sessionStorage.getItem('faveProjects'); return savedProjects ?
JSON.parse(savedProjects) : []; } catch (error) {
console.error('Erro ao recuperar projetos salvos:', error);
return []; }};const saveFaveProjects = (projects) => { try {
sessionStorage.setItem('faveProjects', JSON.stringify(projects));
} catch (error) { console.error('Erro ao salvar projetos
favoritos:', error); }};
```


Conclusão

O uso do Session Storage em aplicações React oferece uma solução elegante para gerenciar estados temporários, como a lista de projetos favoritos em nosso exemplo. Esta abordagem proporciona uma experiência de usuário mais fluida e consistente durante a sessão, sem a necessidade de persistir dados a longo prazo.

Ao implementar o Session Storage, é crucial considerar as limitações e seguir as boas práticas para garantir uma aplicação robusta e eficiente. Com o equilíbrio correto entre persistência de dados e limpeza automática, podemos criar interfaces interativas que respondem às ações do usuário de forma eficaz, mantendo a simplicidade e a performance da aplicação.

Lembre-se sempre de avaliar cuidadosamente quais dados devem ser armazenados no Session Storage e quais requerem uma solução de persistência mais duradoura. Com essa compreensão, você estará bem equipado para criar aplicações React que oferecem uma experiência de usuário superior e gerenciam dados de forma eficiente.

