
otterwise

COMUNICAÇÃO COM A API

otterwise

Comunicação com a API

- Comunicação com a API é essencial na criação de sistemas Web robustos, pois permite uma maior interação de usuário e acesso a informações.
- Mesmo em aplicações estáticas como landings é importante ter algum tipo de interação, como formulários de contato.
- Pacotes de requisições HTTP são o que possibilita a comunicação da aplicação com APIs.
    - Fetch (nativo JS Web | https://www.npmjs.com/package/node-fetch)
    - Axios (https://github.com/axios/axios)

Exemplo

1. Instalação do pacote axios.
2. Realizar a listagem de posts através do enpoint (https://jsonplaceholder.typicode.com/users) utilizando o axios.
3. Configuração em um único local (provider).
4. Organização do código (services).

Exercício

1. Instalação do pacote axios.
2. Realizar a listagem de posts através do enpoint (https://jsonplaceholder.typicode.com/posts) utilizando o axios.
3. Configuração em um único local (provider).
4. Organização do código (services).

NAVEGAÇÃO

otterwise

Navegação

- Quando temos um sistema com múltiplas páginas (home, sobre nós, contato, etc) precisamos de uma configuração de roteamento para informar ao sistema como navegar entre elas.
- Felizmente existem pacotes dedicados a isso.
- Exemplo:
  - **React Router** (https://reactrouter.com).
- Prática:
  - Instalar o pacote.

Rotas

- A aplicação segue sendo uma **SPA**, mas com o **React Router** criam-se **caminhos lógicos** para que seja **simulado** uma troca de página.
- Esses caminhos lógicos são as rotas (ou páginas).
- Prática:
  - Criar o sistema de rotas na aplicação e transformar o componente **App** na **Home**.
  - Criar uma página **Posts** listando os posts da Fake API utilizando o Axios.

TROCANDO INFORMAÇÕES

otterwise

Trocando Informações

- Devido a estrutura gerada pelo React Router nem sempre será possível trocar informação por propriedade como aprendemos.
- As soluções para esse problema é URL parameters ou Search Params

Referências

- https://reactrouter.com/docs/en/v6/getting-started/tutorial#reading-url-params
- https://reactrouter.com/docs/en/v6/getting-started/tutorial#search-params

URL Parameters

- Normalmente utilizado para identificar um recurso.
- Traz um aspecto semântico para as URLs da aplicação semelhante ao Padrão REST.
- Dentro do componente temos acesso à informação utilizando o hook *useParams* (hook customizado do pacote).
- Estrutura: route/:variable
  - Exemplo: posts/5
- Prática:
  - Criar uma página **Post**, acessível com o URL parameter *:id*, carrega as informações de um post (título e mensagem).

Referência: https://reactrouter.com/docs/en/v6/getting-started/tutorial#reading-url-params

Search Parameters

- Parâmetros passados ao final da URL.
- Essas informações são acessadas utilizando o hook customizado *useSearchParams*.
- Normalmente utilizado para filtrar ou ordenar os recursos.
- Estrutura: api-address/rota**?param-one=value-one&param-two=value-two**
  - Exemplo: /posts?userId=1
- Prática:
  - Adicionar informações de search parameters no componente Posts para realizar um filtro de posts por usuário.
- Diferença entre Search Parameters e URL Parameters.

Referência: https://reactrouter.com/docs/en/v6/getting-started/tutorial#search-params

404 - NO MATCH

otterwise

404 - No Match

- Caso o usuário tente acessar uma página que não estamos prevendo, por descuido ou erro de digitação, o sistema deve informar para que o mesmo consiga voltar ao fluxo de navegação.
- Com o React Router podemos configurar um componente para aparecer sempre que isso ocorrer.
- Prática:
  - Adicionar página de 404

Referência: https://stackblitz.com/github/remix-run/react-router/tree/main/examples/basic?file=src/App.tsx

EXERCÍCIO

Exercício

1. Criar um projeto e adicionar roteamento com a react-router-dom.

2. Criar um componente Home que será a rota inicial da aplicação, nela adicionar uma listagem de comentários.

3. Criar um componente Comment que será uma página do comentário com título e descrição.

4. Ao clicar em um comentário na Home, redirecionar para a página de comentário (Comment) com os dados carregados do comentário.

Utilizar a https://jsonplaceholder.typicode.com/ para acessar os recursos de comentários.

LOCALSTORAGE

Bearer Token

- Token de autenticação gerado pelo servidor para identificar o cliente.
- Para acessarmos informações protegidas devemos passar essa informação na requisição.

Local Storage

- O navegador possui um armazenamento local onde as aplicações podem salvar e recuperar informações.
- Essas informações são separadas por host, de forma que uma aplicação não consegue ler o localstorage de outra aplicação.
- Através da aba de aplicação, no navegador, podemos ver e manipular essa informação para facilitar o desenvolvimento e testes.

Métodos

- localstorage.**setItem**(key, value)
  - Cria, ou altera, a entrada *key* com o valor passado em *value* no localstorage.
- localstorage.**getItem**(key)
  - Recupera a informação com chave *key* do localstorage.
- localstorage.**removeItem**(key)
  - Remove a entrada *key* do localstorage.

Exemplo

1. Realizar um login fake e salvar token no localstorage.
2. Adicionar interceptor ao *provider* com o token nas requisições do axios.