React

Prof. José Matheus

O que é Vite?

Vite é uma ferramenta de construção (build tool) rápida e eficiente para o desenvolvimento de aplicações web modernas. Ele é especialmente adequado para projetos usando frameworks como React, Vue.js, e Svelte. Vite se destaca pela sua inicialização rápida, recarregamento rápido do navegador (hot module replacement), e suporte nativo a módulos ES, permitindo que você desenvolva sua aplicação sem a necessidade de transpilação de código.

Vantagens do Vite

Algumas vantagens do Vite em comparação com abordagens de criação tradicionais e o Create React App (**CRA**)

- **Inicialização rápida:** O Vite é rápido para iniciar projetos devido à sua compilação sob demanda;
- **Tempo de desenvolvimento ágil:** Atualizações instantâneas e recarregamento rápido de módulos tornam o desenvolvimento mais eficiente;
- **Suporte nativo a ES Modules:** O Vite aproveita o suporte nativo a módulos ES no navegador, simplificando o desenvolvimento e eliminando a necessidade de transpilação;
- **Ecossistema moderno:** Suporta tecnologias modernas como TypeScript, Vue.js e React, com integração fácil;
- **Configuração simples:** A configuração padrão é mínima e fácil de entender;
- **Eco-friendly:** Consome menos recursos do sistema durante o desenvolvimento, tornando-o mais sustentável.

Inicializando um projeto com Vite

Para inicializar um projeto ReactJS com Vite, você pode seguir os seguintes passos:

- Certifique-se que o **Node.js** está instalado em sua máquina;
- Digite o seguinte comando no terminal: `npm init vite`.Substitua **vite-project** pelo nome desejado para o seu projeto, depois escolha a opção “**React**” e depois “**JavaScript**”.
- Depois de escolher dentre as opções apresentadas pelo Vite, digite o seguinte comando no terminal: `cd nome-do-meu-projeto`;
- Em seguida, você precisa instalar as dependências do projeto. Execute o seguinte comando: `npm install`. Depois disso, é só inicializar o servidor com o comando `npm run dev`.

Django

Django CORS

Antes de inicializarmos nosso font-end com React, devemos fazer algumas alterações no nosso código do Django.

O Django nativamente bloqueia o CORS (Cross-Origin Resource Sharing) por razões de segurança. CORS é um mecanismo de segurança que impede que scripts em uma página da web façam solicitações a um domínio diferente do que a página está atualmente carregada. Isso é uma proteção contra ataques de origem cruzada.

Ao desenvolver uma aplicação web que utiliza o Django como backend e o React como frontend, é provável que o frontend (React) esteja sendo servido em um domínio ou porta diferente do backend (Django). Por exemplo, o backend pode estar sendo executado em localhost:8000, enquanto o frontend é servido em localhost:3000.

Django CORS

Para permitir que o frontend (ou qualquer outra aplicação) faça solicitações ao backend, nós precisamos configurar o CORS no Django. O pacote **django-cors-headers** facilita essa configuração.

Devemos então instalar essa dependência em nosso projeto django: pip install django-cors-headers
Depois, temos que fazer algumas alterações em nosso arquivo **settings.py**

JSON

```
CORS_ORIGIN_ALLOW_ALL = True
ALLOWED_HOSTS = ['*']

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    ...
]
```

Django CORS

- **CORS_ORIGIN_ALLOW_ALL = True:** Essa configuração permite que todas as origens (ou seja, todos os domínios) acessem seu backend. Isso é útil durante o desenvolvimento, mas em um ambiente de produção, você pode querer especificar os domínios permitidos explicitamente;
- **ALLOWED_HOSTS = ['*']:** Isso permite que seu servidor Django aceite solicitações de qualquer host. No entanto, em um ambiente de produção, é uma prática recomendada especificar os hosts permitidos explicitamente;
- **MIDDLEWARE:** Adiciona o middleware do corsheaders ao pipeline de middleware do Django, garantindo que as solicitações recebam os cabeçalhos CORS apropriados.

Essas configurações permitem que o frontend React faça solicitações ao backend Django sem ser bloqueado pelo CORS. Certifique-se de ajustar essas configurações conforme necessário, especialmente em um ambiente de produção, para garantir a segurança e a integridade da sua aplicação.

React

React: Dependências

Antes de tudo, devemos antes instalar as dependências que iremos usar no nosso projeto:

```
npm install axios --save
```

```
npm install react-router-dom --save
```

O NPM fornece a opção *--save* ao instalar pacotes. Se usarmos a opção *--save* após instalar o pacote, ele será salvo no arquivo **package.json** dentro das dependências.

React: Variáveis globais

Para configurar variáveis globais no React, podemos usar o contexto ou criar um arquivo de configuração que exporta as variáveis globais. Vamos criar um arquivo **config.js** para armazenar a URL da API:

JSX

```jsx
// src/config.jsx
const API_URL = 'http://localhost:8000/api'

export { API_URL }
```

Componente para consumir a API

Agora, vamos criar componentes React para consumir a API Django. Primeiramente, criamos uma pasta chamada "**components**" dentro da pasta **src**. Depois, vamos criar um componente **PostList** para exibir uma lista de posts.

- **Importações:** Importamos useState e useEffect do React para gerenciar o estado e o ciclo de vida do componente. Também importamos o axios para fazer solicitações HTTP e a API_URL de um arquivo de configuração.
- **Componente PostList:** Definimos o componente PostList como uma função.
- **Estado de posts:** Utilizamos o hook useState para criar um estado posts inicializado como um array vazio.
- **Efeito useEffect:** Usamos o hook useEffect para realizar operações de efeito colateral, como chamadas de API. Ele é executado após a renderização inicial do componente ([] como segundo argumento). Dentro do efeito, fazemos uma solicitação GET para API_URL/posts/ usando o axios. Quando a resposta é recebida com sucesso, atualizamos o estado posts com os dados recebidos. Se houver algum erro, registramos o erro no console.
- **Renderização:** Na renderização do componente, exibimos uma lista de posts em um <ul>. Usamos map para iterar sobre o array posts e renderizar cada post como um <li>. Para cada post, exibimos o título, conteúdo e imagem, usando dados do objeto post.
- **Exportação:** Exportamos o componente PostList para que possa ser utilizado em outros lugares da aplicação.

```jsx
// src/components/PostList.jsx
import { useState, useEffect } from 'react'
import axios from 'axios'
import { API_URL } from '../config'

function PostList() {
  const [posts, setPosts] = useState([])

  useEffect(() => {
    axios.get(`${API_URL}/posts/`)
      .then(response => {
        setPosts(response.data)
      })
      .catch(error => {
        console.error('Erro ao buscar posts: ', error)
      })
  }, [])

  return (
    <div>
      <h1>Lista de Posts</h1>
      <ul>
        {posts.map(post => (
          <li key={post.id}>
            <h2>{post.title}</h2>
            <p>{post.content}</p>
            <img src={post.image} alt={post.title} />
          </li>
        ))}
      </ul>
    </div>
  )
}

export default PostList
```

Componente para consumir a API

Agora, vamos consumir o componente **PostList** no componente principal **App**.

```
// src/App.jsx
import PostList from
'./components/PostList'

function App() {
  return (
    <div>
      <h1>Minha Aplicação
React com API Django</h1>
      <PostList />
    </div>
  )
}

export default App
```

React CRUD

Componentes do CRUD **SEM** autenticação

Componentes do CRUD **COM** autenticação

Dúvidas?

Referências

React. Disponível em: <https://react.dev/>