

Análise e Desenvolvimento de Sistemas

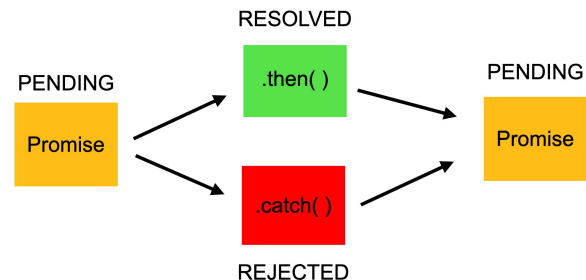
Frameworks Web I

Aula 05 - React JS: parte 4



Promise

Promise



Uma Promise em JavaScript é um objeto que representa um valor que pode estar disponível agora, no futuro ou nunca.

- É frequentemente usado para lidar com operações assíncronas, como carregar dados de uma API ou ler um arquivo, onde não sabemos exatamente quando o resultado estará pronto.
- A Promise permite que você trate essas operações de forma mais organizada, fornecendo um mecanismo para lidar com sucesso ou falha quando os dados estiverem disponíveis.
- Por exemplo, você pode usar `.then()` para lidar com o sucesso e `.catch()` para tratar erros quando a operação assíncrona for concluída, tornando o código mais legível e eficiente.

Fetch

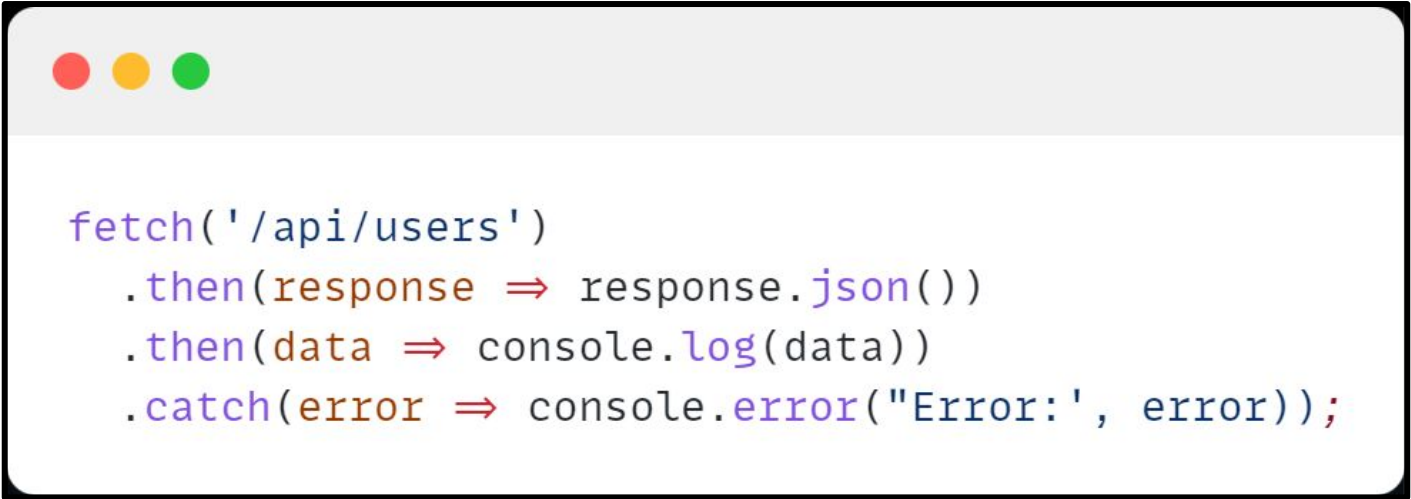
Fetch

A API Fetch é um mecanismo moderno baseado em promessas (promises) para fazer solicitações HTTP no navegador

- Ele está integrado na maioria dos navegadores modernos e é uma ferramenta poderosa para fazer solicitações e lidar com respostas.
- Na sua forma mais simples, você fornece uma URL para a função fetch e ela retorna uma promise que é resolvida para o objeto Response que representa a resposta à solicitação.
- Este objeto Response inclui o status da solicitação, cabeçalhos e métodos para processar o corpo da resposta.

Fetch

Abaixo é mostrado um exemplo simples de utilização:



```
fetch('/api/users')  
  .then(response => response.json())  
  .then(data => console.log(data))  
  .catch(error => console.error("Error:", error));
```

Fetch

Neste exemplo, estamos buscando uma lista de usuários de uma API

- A função `fetch` retorna uma promessa, com a qual trataremos com `then`.
- O primeiro `then` recebe um objeto `Response`. Chamamos o método `json` no objeto `Response`, que por sua vez retornará uma outra promessa que devolverá os dados convertidos quando estiverem prontos.
- O segundo `then` trata os dados processados pelo método `json`.
- Se ocorrer um erro em qualquer ponto, o `catch` irá lidar com isso.

Fetch

A API Fetch suporta todos os métodos HTTP, não apenas GET. Para fazer uma solicitação POST, por exemplo, você pode passar um objeto de opções como segundo argumento para a função fetch:



```
fetch('/api/users', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Tilápia',
    email: 'tilapia@gmail.com',
  }),
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

Fetch

Aqui, estamos enviando uma carga JSON ao servidor para criar um novo usuário.

- A propriedade `method` especifica o método HTTP.
- A propriedade `headers` é um objeto que contém todos os cabeçalhos que desejamos incluir na solicitação.
- Já a propriedade `body` é o corpo da solicitação.

Axios

Axios

O Axios é uma biblioteca JavaScript que auxilia os desenvolvedores a realizar operações de comunicação com a internet, como buscar informações de um servidor web ou enviar dados para um servidor remoto.

- Você pode pensar no Axios como um mensageiro digital ou serviço de correio que seu programa utiliza para enviar ou receber dados através da internet.
- Em vez de cartas ou pacotes físicos, o Axios lida com informações digitais que você deseja buscar ou transmitir pela web, tornando a interação entre seu aplicativo e os servidores web mais eficiente e acessível.
- Para instalá-lo no projeto, use:

```
npm install axios
```

Axios

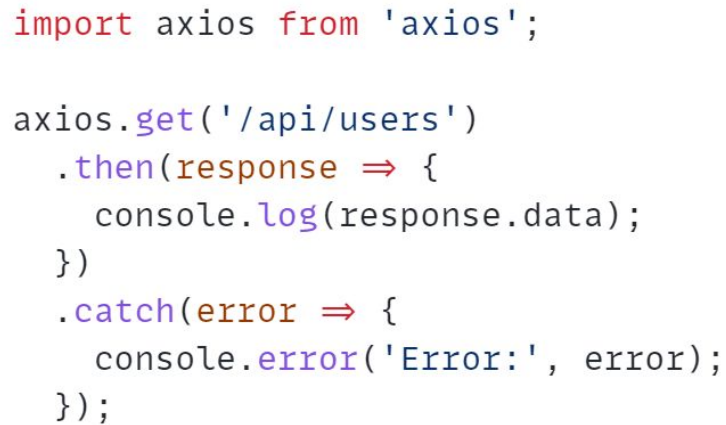
Pontos-chave sobre o Axios:

- **Facilidade de Uso:** O Axios é uma biblioteca muito fácil de usar. Ele fornece uma maneira simples de criar e enviar solicitações para servidores web a partir do seu código JavaScript.
- **Compatibilidade:** O Axios é compatível com a maioria dos navegadores da web e ambientes de desenvolvimento JavaScript, tornando-o uma escolha popular para trabalhar com solicitações HTTP em diferentes plataformas.
- **Promisses:** O Axios usa Promisses, que são uma maneira de lidar com operações assíncronas em JavaScript. Isso significa que você pode fazer uma solicitação para um servidor e, quando os dados estiverem prontos, você receberá uma resposta, tudo de forma assíncrona.
- **Versatilidade:** O Axios pode ser usado para várias finalidades, como buscar dados de APIs, enviar dados de formulários, fazer upload de arquivos, realizar autenticação em aplicativos e muito mais.
- **Tratamento de Erros:** Ele também lida bem com erros, fornecendo informações úteis sobre o que deu errado em caso de problemas durante uma solicitação.

Get

Axios

Exemplo de requisição GET com Axios



```
import axios from 'axios';

axios.get('/api/users')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Axios

No exemplo anterior, estamos enviando uma solicitação GET para o endpoint `'/api/users'`.

- O método `get` retorna uma promessa que é resolvida com a resposta da solicitação.
- Se a solicitação for bem-sucedida, o bloco `then` será executado, e se ocorrer um erro, o bloco `catch` lidará com ele.

Post

Axios

Exemplo de requisição POST com Axios



```
import axios from 'axios';

axios.post('api/users', {
  name: 'Tilápia',
  email: 'tilapia@gmail.com'
})
.then(response => {
  console.log(response.data);
})
.catch(error => {
  console.error('Error:' , error);
});
```

Axios

No exemplo acima, o segundo argumento do método `post` contém os dados que desejamos enviar no corpo da solicitação.

- O Axios realiza automaticamente a conversão desses dados em formato JSON para nós.
- Isso significa que podemos enviar informações de maneira simples, e o Axios cuida de transformá-las no formato apropriado para a solicitação.

Interceptadores e CancelToken

Axios

Outra vantagem significativa do Axios é a capacidade de usar interceptadores. Os interceptadores permitem que você altere globalmente a solicitação ou a resposta antes que elas sejam tratadas pelo bloco then ou catch.

- Por exemplo, você pode configurar um interceptor para adicionar um cabeçalho de Autorização (Authorization header) a todas as solicitações.
- Neste exemplo, antes de cada solicitação ser enviada, estamos adicionando um cabeçalho de Autorização que inclui um token armazenado localmente.

```
axios.interceptors.request.use(config => {  
  config.headers.Authorization = `Bearer ${localStorage.getItem('token')}`;  
  return config;  
});
```

Axios

A capacidade de cancelar solicitações é outra característica poderosa do Axios.

- Ao usar um CancelToken, você pode cancelar uma solicitação. Isso é útil em cenários nos quais um usuário pode navegar para longe de uma página antes que uma solicitação seja concluída.

```
import axios from 'axios';

const CancelToken = axios.CancelToken;
const source = CancelToken.source();

axios.get('/api/users', {
  cancelToken: source.token
}).catch(function (thrown) {
  if (axios.isCancel(thrown)) {
    console.log('Requisição cancelada', thrown.message);
  } else {
    // trata erro
  }
});

// cancela a requisição
source.cancel('Operação cancelada pelo usuário');
```

Axios

Para que serve o cancelToken?

- Imagine que você está fazendo uma solicitação de rede, como buscar dados de um servidor ou fazer upload de um arquivo, e, de repente, o usuário decide cancelar essa operação.
- O CancelToken permite que você cancele essa solicitação em progresso para economizar recursos e melhorar a experiência do usuário.

Axios

Onde é usado o CancelToken:

- O CancelToken é geralmente usado em situações em que você deseja fornecer ao usuário a capacidade de cancelar uma operação que está em andamento. Alguns cenários comuns incluem:
 - **Navegação de Página:** Se o usuário estiver em uma página da web e iniciar uma solicitação de rede, mas decidir navegar para outra página antes que a solicitação seja concluída, você pode usar o CancelToken para cancelar a solicitação automaticamente.
 - **Pesquisa de AutoCompletar:** Em um campo de pesquisa que exibe sugestões à medida que o usuário digita, você pode usar o CancelToken para cancelar solicitações de sugestões anteriores se o usuário continuar digitando rapidamente.
 - **Upload de Arquivos:** Se o usuário começar a fazer upload de um arquivo grande, mas decidir cancelar o upload, o CancelToken permite que você interrompa a operação de upload.

Custom Hook com Axios

Axios

Vamos criar um hook personalizado que usa a biblioteca Axios para realizar uma solicitação GET.

```
import { useState, useEffect } from 'react';
import axios from 'axios';

const useFetch = (url) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await axios.get(url);
        setData(res.data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
};

export default useFetch;
```

Axios

No código anterior começamos importando `useState` e `useEffect` do React, bem como o `axios`, que são os fundamentais do nosso hook personalizado.

- Definimos a função `useFetch`, que recebe uma URL como parâmetro.
- Utilizamos o hook `useState` para configurar três estados essenciais:
 - **`data`**: que armazena os dados da resposta da solicitação GET
 - **`loading`**: valor booleano que indica se os dados ainda estão sendo carregados.
 - **`error`**: para guardar informações sobre qualquer erro que ocorra durante a solicitação.

Axios

- Dentro do `useEffect`, criamos uma função assíncrona chamada `fetchData` que lida com a solicitação GET. Essa função é imediatamente invocada dentro do `useEffect`.
- Utilizamos um bloco `try...catch...finally` para gerenciar a solicitação GET e possíveis erros.
 - No bloco `try`, efetuamos a solicitação GET usando `axios.get(url)` e atualizamos o estado `data` com os dados da resposta.
 - Se ocorrer um erro, o bloco `catch` atualiza o estado `error` com a mensagem de erro correspondente.
 - O bloco `finally` é executado após a conclusão do `try` ou `catch` e define `loading` como `false`, indicando que os dados terminaram de ser carregados.

Axios

A matriz de dependência do `useEffect` inclui `url`, garantindo que `fetchData` seja chamado novamente sempre que `url` for alterado.

Por fim, `useFetch` retorna um objeto com `data`, `loading` e `error`, que podem ser utilizados por componentes que chamam esse gancho.

Axios

Neste componente, simplesmente importamos e chamamos o `useFetch` com uma URL e usamos os dados retornados, o estado de carregamento (`loading`), e os erros para renderizar uma interface de usuário diferente com base no status da solicitação.

Agora, qualquer componente pode utilizar esse gancho personalizado para buscar dados com o Axios:

```
import React from 'react';
import useFetch from './useFetch';

const ComponenteExemplo = () => {
  const { data, loading, error } = useFetch('https://api.example.com/data');

  if (loading) return <div>Carregando...</div>;
  if (error) return <div>Erro: {error}</div>;
  return <div>Dados: {JSON.stringify(data)}</div>;
};

export default ComponenteExemplo;
```

Axios vs Fetch

Axios

Sintaxe e Usabilidade:

- **Axios:** O Axios fornece uma API mais amigável e fácil de usar em comparação com o fetch. Ele retorna promessas com respostas diretas e lida automaticamente com a serialização e desserialização de dados.
- **fetch:** O fetch é uma API nativa do navegador e é mais primitiva em termos de sintaxe. Requer um pouco mais de trabalho manual para lidar com promessas e converter os dados da resposta.

Axios

Suporte a JSON:

- **Axios:** O Axios faz a desserialização de JSON automaticamente, então você pode trabalhar com os dados da resposta diretamente como objetos JavaScript.
- **fetch:** Com o fetch, você precisa chamar `response.json()` explicitamente para desserializar os dados JSON da resposta.

Axios

Tratamento de Erros:

- **Axios:** O Axios possui um sistema de interceptores que facilita o tratamento global de erros em todas as solicitações. Você pode configurar interceptores para manipular erros de uma maneira mais organizada.
- **fetch:** O fetch requer que você lide manualmente com erros, o que pode ser mais complicado em comparação com o Axios.

Axios

Suporte a Cancelamento de Solicitações:

- **Axios:** O Axios possui suporte integrado para cancelamento de solicitações usando `CancelToken`, o que é útil para situações em que o usuário deseja cancelar uma solicitação em andamento.
- **fetch:** O `fetch` não possui um mecanismo integrado para cancelamento de solicitações, e você deve usar a API `AbortController` para alcançar funcionalidade semelhante.

Axios

Compatibilidade do Navegador:

- **Axios:** O Axios funciona tanto em navegadores quanto em ambientes Node.js. Ele fornece um mecanismo de adaptação para trabalhar em ambos os ambientes.
- **fetch:** O fetch é nativo do navegador e requer um polyfill ou adaptações adicionais para ser usado em ambientes Node.js.
 - Um polyfill é um pedaço de código que permite adicionar funcionalidades modernas em navegadores mais antigos que não suportam essas funcionalidades nativamente, garantindo que aplicativos da web funcionem de maneira consistente em diferentes ambientes.

Axios

Ambos Axios e fetch têm seu lugar e podem ser usados com sucesso em projetos React.

A escolha entre eles depende das necessidades do projeto e da preferência pessoal. Se você precisa de uma API mais amigável e fácil de usar com recursos adicionais, o Axios pode ser a escolha certa. Se você deseja algo mais leve e nativo do navegador, o fetch é uma boa opção.

Tratamento de erros

Tratamento de erros

O tratamento de erros é um aspecto crítico da programação.

- Em um mundo perfeito, tudo funcionaria como esperado, e não precisaríamos nos preocupar com erros.
- No entanto, no mundo real, as coisas podem, e frequentemente, dão errado. Portanto, ter um entendimento sólido de como lidar com erros pode melhorar significativamente a robustez de sua aplicação.

Tratamento de erros

No React, existem dois tipos de erros a serem considerados: erros de JavaScript e erros de tempo de execução.

- Erros de JavaScript podem ocorrer durante o processo de renderização, nos métodos do ciclo de vida de um componente ou no construtor de uma classe.
- Esses erros podem fazer com que toda a árvore de componentes seja desmontada se não forem tratados adequadamente, o que resulta em uma experiência do usuário ruim.

Tratamento de erros

No React, existem ferramentas chamadas "Limites de Erro" (Error Boundaries) que ajudam a lidar com erros de JavaScript.

- Quando algo sai errado em um componente ou em qualquer lugar dentro dele, em vez de mostrar um erro ao usuário, o React usa esses Limites de Erro para pegar o erro, registrar o que aconteceu e exibir uma página alternativa em vez de mostrar o erro real.
- Isso melhora a experiência do usuário, pois o aplicativo não quebra completamente quando um erro ocorre. Esses Limites de Erro podem pegar erros que acontecem durante a exibição de informações, nas etapas do ciclo de vida do componente e até mesmo nos estágios iniciais de criação dos componentes. Eles tornam o aplicativo mais robusto e tolerante a falhas.

Tratamento de erros

O tratamento de erros por Error Boundary nativo do React JS só está disponível para implementações por componentes de classe.

- Para utilizar esta estratégia com componentes funcionais, é necessário usar uma biblioteca externa chamada: react-error-boundary
- Para instalá-la use:

```
npm install react-error-boundary
```

Tratamento de erros

Exemplo

Neste exemplo, usamos o componente `ErrorBoundary` do `react-error-boundary` para envolver `GeraErro`. Se ocorrer um erro em `GeraErro`, o `ErrorBoundary` renderizará o componente `ErrorFallback`, que exibe uma mensagem de erro amigável.

Usar o `react-error-boundary` simplifica muito a implementação do Error Boundary em componentes de função, tornando a manipulação de erros mais fácil e eficaz.

```
import React from 'react';
import { ErrorBoundary } from 'react-error-boundary';

function GeraErro() {
  throw new Error('Este é um erro intencional');
}

function ErrorFallback({ error }) {
  return (
    <div>
      <h1>Algo deu errado!</h1>
      <p>{error.message}</p>
    </div>
  );
}

function App() {
  return (
    <div>
      <h1>Exemplo de Error Boundary com react-error-boundary</h1>
      <ErrorBoundary FallbackComponent={ErrorFallback}>
        <GeraErro />
      </ErrorBoundary>
    </div>
  );
}

export default App;
```

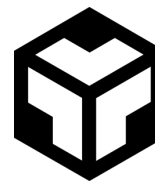
Tratamento de erros

Importante

- O React apenas lida com erros lançados durante a renderização ou durante os métodos do ciclo de vida do componente. Erros lançados em manipuladores de eventos ou após a execução de código assíncrono não serão capturados.
- Isso pode ser tratado usando os mecanismos nativos de tratamento de erros do JavaScript, como blocos try/catch.

Prática

Prática



[Acesse o projeto](#)

Nesta prática vamos utilizar alguns conceitos aprendidos até aqui para construir uma aplicação simples que se comunique com uma API :)

- Ao final da implementação, este será o resultado

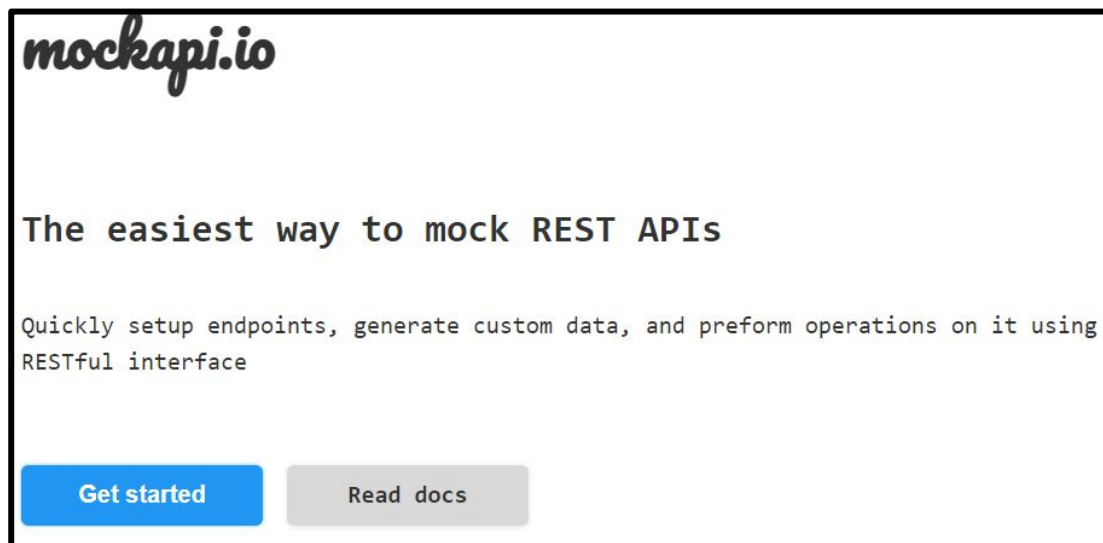


Criando a API

Prática

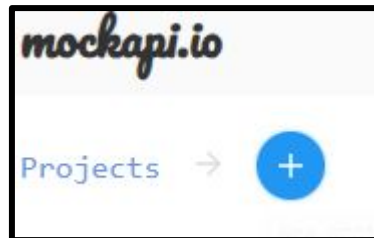
Para criar a API vamos usar a ferramenta “Mock API”, que permite a criação rápida de uma API pronta para testes :)

- Acesse: <https://mockapi.io>
- Clique em “Get Started” e crie uma conta na plataforma.



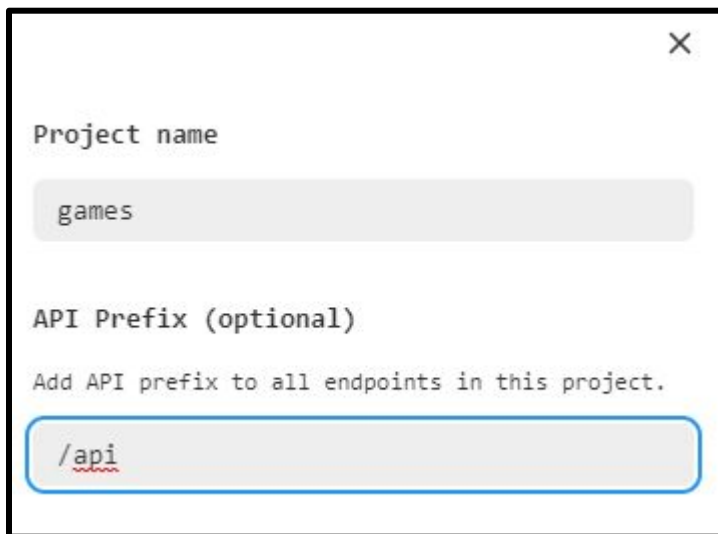
Prática

Crie um novo projeto, para isso, clique em “+”



Prática

Dê um nome para o projeto e adicione o prefixo /api. Por fim, clique em “Create



Project name

games

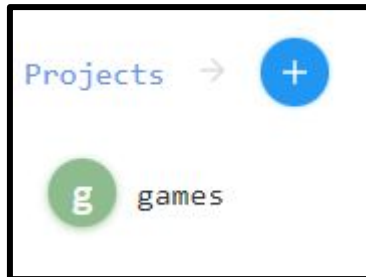
API Prefix (optional)

Add API prefix to all endpoints in this project.

/api

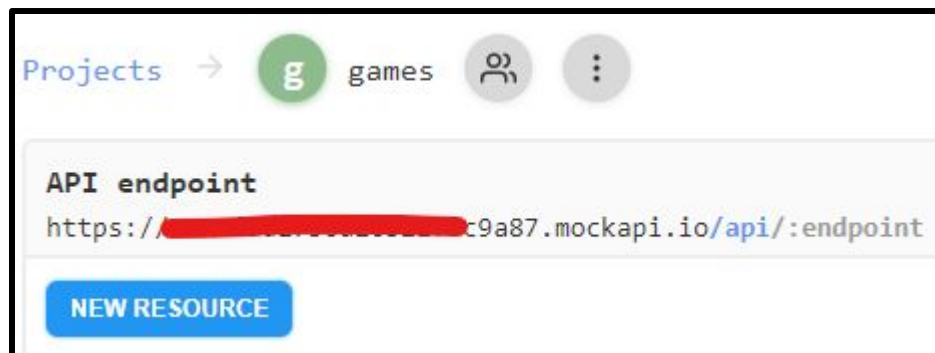
Prática

Abra o projeto clicando nele



Prática

Este será o endereço a ser utilizado no seu código



Prática

Clique em “New Resource” para configurar os campos da API.

Por fim clique em “Create”

Resource name

Enter meaningful resource name, it will be used to generate API endpoints.

games

Schema (optional)

Define Resource schema, it will be used to generate mock data.

id	Object ID
title	STRING
platform	STRING

Criação do projeto

Prática

Criando e abrindo o projeto

- Crie um novo projeto, chamarei ele de “game-collection”, para isso execute no terminal o seguinte comando:

```
npm create vite@latest game-collection -- --template react
```

- Em seguida, execute os comandos abaixo:

```
cd game-collection  
npm install  
npm run dev
```

Prática

Para o projeto "GameCollection" com React, Material-UI e Axios, você precisará instalar as seguintes dependências:

- Material-UI: Uma biblioteca de componentes React com design e estilos prontos para uso.
- Axios: Uma biblioteca para fazer requisições HTTP.
- Você também pode instalar ícones do Material-UI para melhorar a aparência do projeto.

Use o seguinte comando para instalar essas dependências no seu projeto:

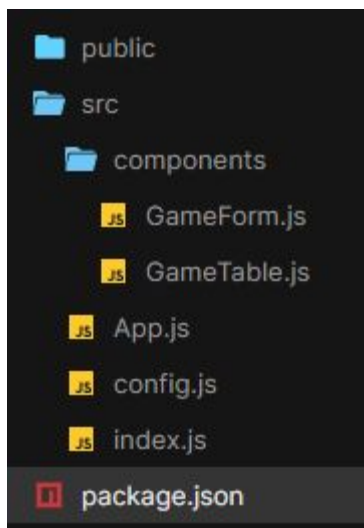
```
npm install @mui/material @mui/icons-material @emotion/react  
@emotion/styled axios
```


Organizando os arquivos

Prática

Para melhor organizar os arquivos, crie, dentro da pasta src uma outra pasta chamada “components”.

- Ao final, a estrutura de seu projeto estará assim:



Configuração da API

Prática

Crie um arquivo chamado `config.js` na raiz do projeto (mesmo nível que a pasta `src`). Defina a URL da API que será usada no projeto:

A code editor window with a light gray header bar containing three colored window control buttons (red, yellow, green). The main area is white and contains the following JavaScript code:

```
// config.js
const API_URL = 'https://sua-api-aqui.com'; // Substitua pelo URL da sua API
export default API_URL;
```

```
// config.js
const API_URL = 'https://sua-api-aqui.com'; // Substitua pelo URL da sua API
export default API_URL;
```

Componente App

Prática

Arquivo App.js

Esse arquivo é o ponto de entrada principal do aplicativo e contém o componente App no projeto.



[Clique aqui para ampliar a imagem](#)

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import GameForm from "../components/GameForm";
import GameTable from "../components/GameTable";
import {
  CssBaseline,
  Container,
  Typography,
  AppBar,
  Toolbar
} from "@mui/material";
import API_URL from "../config";

const appBarStyle = {
  marginBottom: "20px"
};

const pageTitleStyle = {
  fontSize: "2rem",
  fontWeight: "bold",
  marginBottom: "20px"
};

function App() {
  const [games, setGames] = useState([]);
  const [showForm, setShowForm] = useState(false);

  useEffect(() => {
    fetchGames();
  }, []);

  const fetchGames = async () => {
    try {
      const response = await axios.get(`${API_URL}/games`);
      setGames(response.data);
    } catch (error) {
      console.error("Erro ao buscar jogos:", error);
    }
  };

  const handleAddGame = async (newGame) => {
    try {
      await axios.post(`${API_URL}/games`, newGame);
      fetchGames();
      setShowForm(false);
    } catch (error) {
      console.error("Erro ao adicionar jogo:", error);
    }
  };

  const handleDeleteGame = async (gameId) => {
    try {
      await axios.delete(`${API_URL}/games/${gameId}`);
      fetchGames();
    } catch (error) {
      console.error("Erro ao excluir jogo:", error);
    }
  };

  return (
    <div>
      <CssBaseline />
      <AppBar position="static" style={appBarStyle}>
        <Toolbar>
          <Typography variant="h6">Coleção de Jogos</Typography>
        </Toolbar>
      </AppBar>
      <Container maxWidth="lg">
        <Typography variant="h2" align="center" style={pageTitleStyle}>
          Coleção de Jogos
        </Typography>
        {showForm ? (
          <GameForm handleAddGame={handleAddGame} setShowForm={setShowForm} />
        ) : (
          <GameTable
            games={games}
            handleDeleteGame={handleDeleteGame}
            setShowForm={setShowForm}
          />
        )}
      </Container>
    </div>
  );
}

export default App;
```

Componente Index

Prática

Arquivo Index.js

É o ponto de entrada para a renderização do aplicativo React no projeto.



[Clique aqui para ampliar a imagem](#)

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";

import App from "./App";

const rootElement = document.getElementById("root");
const root = createRoot(rootElement);

root.render(
  <StrictMode>
    <App />
  </StrictMode>
);
```


Componente GameForm

Prática

Arquivo GameForm.js

Este componente é responsável por exibir o formulário de adição de jogos.



[Clique aqui para ampliar a imagem](#)

```
import React, { useState } from "react";
import { TextField, Button, Grid, Paper, Typography } from "@mui/material";

const formStyle = {
  padding: "16px",
  maxWidth: "400px",
  margin: "auto"
};

const buttonStyle = {
  marginRight: "8px"
};

function GameForm({ handleAddGame, setShowForm }) {
  const [newGame, setNewGame] = useState({ title: "", platform: "" });

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setNewGame({ ...newGame, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    handleAddGame(newGame);
    setNewGame({ title: "", platform: "" });
  };

  return (
    <Paper elevation={3} style={formStyle}>
      <Typography variant="h6" gutterBottom>
        Adicionar Jogo
      </Typography>
      <Form onSubmit={handleSubmit}>
        <Grid container spacing={2}>
          <Grid item xs={12}>
            <TextField
              fullWidth
              label="Título"
              name="title"
              value={newGame.title}
              onChange={handleInputChange}
            />
          </Grid>
          <Grid item xs={12}>
            <TextField
              fullWidth
              label="Plataforma"
              name="platform"
              value={newGame.platform}
              onChange={handleInputChange}
            />
          </Grid>
        </Grid>
        <div style={{ marginTop: "16px" }}>
          <Button
            variant="contained"
            color="primary"
            type="submit"
            style={buttonStyle}
          >
            Adicionar
          </Button>
          <Button onClick={() => setShowForm(false)} style={buttonStyle}>
            Cancelar
          </Button>
        </div>
      </Form>
    </Paper>
  );
}

export default GameForm;
```

Componente GameTable

Prática

Arquivo GameTable.js

Este componente é responsável por exibir a tabela de jogos e a funcionalidade de exclusão.



[Clique aqui para ampliar a imagem](#)

```
import React, { useState } from "react";
import {
  Button,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  Paper,
  Typography,
  Box,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
} from "@mui/material";
import DeleteIcon from "@mui/icons-material/Delete";
import AddCircleOutlineIcon from "@mui/icons-material/AddCircleOutline";

const tableCellStyle = {
  width: 600px,
  margin: "auto",
  marginTop: "20px"
};

const headerCellStyle = {
  backgroundColor: "#f5f5f5",
  fontWeight: "bold"
};

function GameTable({ games, handleDeleteGame, setShowForm }) {
  const [openDialog, setOpenDialog] = useState(false);
  const [gameToDelete, setGameToDelete] = useState(null);

  const handleConfirmDelete = () => {
    if (gameToDelete) {
      handleDeleteGame(gameToDelete.id);
      setGameToDelete(null);
    }
    setOpenDialog(false);
  };

  const handleOpenDialog = (game) => {
    setGameToDelete(game);
    setOpenDialog(true);
  };

  return (
    <div>
      <Box display="flex" justify-content="space-between" align-items="center">
        <Typography variant="h2">Lista de Jogos</Typography>
        <Button
          variant="contained"
          color="primary"
          startIcon={AddCircleOutlineIcon}
          onClick={() => setShowForm(true)}
        >
          Adicionar Jogo
        </Button>
      </Box>
      <TableContainer component={Paper} style={tableCellStyle}>
        <Table>
          <TableHead>
            <TableRow>
              <TableCell style={headerCellStyle} align="center">
                Titulo
              </TableCell>
              <TableCell style={headerCellStyle} align="center">
                Plataforma
              </TableCell>
              <TableCell style={headerCellStyle} align="center">
                Anos
              </TableCell>
            </TableRow>
          <TableHead>
            <TableBody>
              {games.length === 0 ? (
                <TableRow>
                  <TableCell colspan={3} align="center">
                    <Typography variant="subheading">
                      Não há jogos disponíveis.
                    </Typography>
                  </TableCell>
                </TableRow>
              ) : (
                games.map((game) => (
                  <TableRow key={game.id}>
                    <TableCell align="center">{game.title}</TableCell>
                    <TableCell align="center">{game.platform}</TableCell>
                    <TableCell align="center">
                      <Button
                        variant="outlined"
                        color="error"
                        startIcon={DeleteIcon}
                        onClick={() => handleOpenDialog(game)}
                      >
                        Excluir
                      </Button>
                    </TableCell>
                  </TableRow>
                )
              )
            </TableBody>
          </Table>
        </TableContainer>

        <Dialog open={openDialog} onClose={() => setOpenDialog(false)}>
          <DialogTitle>Confirmar Exclusão</DialogTitle>
          <DialogContent>
            Tem certeza de que deseja excluir o jogo "{gameToDelete.title}"?
          </DialogContent>
          <DialogActions>
            <Button onClick={() => setOpenDialog(false)} color="primary">
              Cancelar
            </Button>
            <Button onClick={handleConfirmDelete} color="error">
              Confirmar
            </Button>
          </DialogActions>
        </Dialog>
      </div>
    </div>
  );
}

export default GameTable;
```

Execute e teste

Prática

Inicie o servidor de desenvolvimento e abra o aplicativo no navegador:

```
npm start
```

- Você verá a página inicial do "GameCollection" com a lista de jogos (ou a mensagem "Não há jogos disponíveis" se a lista estiver vazia).
- Clique no botão "Adicionar Jogo" para abrir o formulário de adição de jogos. Preencha o título e a plataforma do jogo e clique em "Adicionar" para adicionar um novo jogo. Você pode excluir jogos clicando no botão "Excluir" na tabela e confirmando a exclusão na caixa de diálogo de confirmação.

Sua Vez

Sua Vez

O que fazer:

- Com base no projeto anterior, proponha e crie uma nova API no “Mock API” para um problema diferente do mostrado.
- Crie um projeto para manipulação desta API, com as operações de Listagem (Get), Inserção (POST) e Remoção (DELETE) dos dados.
- Suba o seu projeto para a plataforma CodeSandbox: <https://codesandbox.io>
- Envie um vídeo curto mostrando o projeto executando e o link de compartilhamento da plataforma na atividade do ciclo 5 no Moodle.

Extra

Extra



[Acesse o projeto](#)

Projeto ToDo com API :)

✓ Lista de Tarefas

Nome da Tarefa

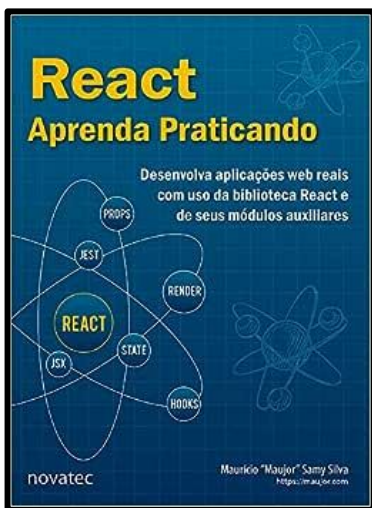
Baixa Prioridade

+ Adicionar Tarefa

Tarefa	Prioridade	Ações
teste222werasd	Média	<button>Remover</button> <button>Atualizar</button>
tesdfsdfs	Baixa	<button>Remover</button> <button>Atualizar</button>

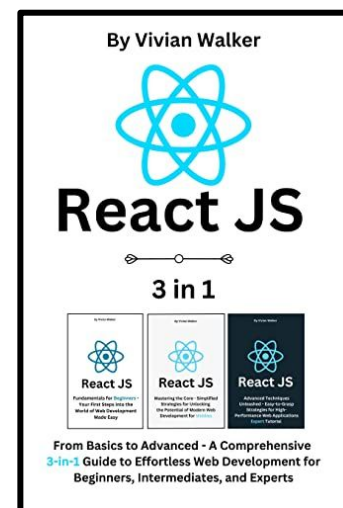
Referências

Referências



SILVA, Maurício Samy. **React - Aprenda praticando**. Novatec. 2021.

Walker, Vivian. **React JS: From Basics to Advanced - A Comprehensive 3-in-1 Guide to Effortless Web Development for Beginners, Intermediates, and Experts**.



Análise e Desenvolvimento de Sistemas

Frameworks Web I

Aula 05 - React JS: parte 4
