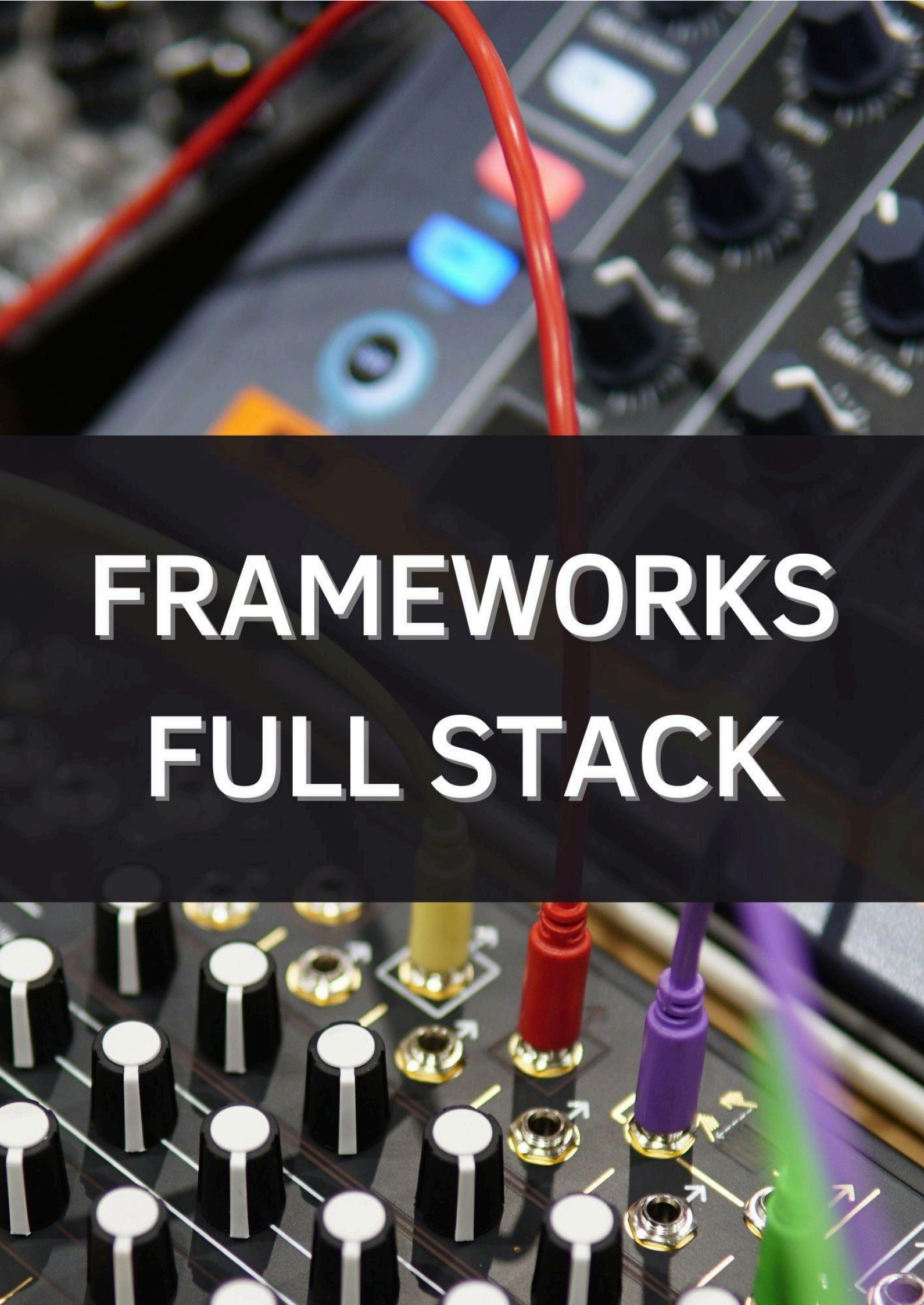

FRAMEWORKS FULL STACK

Texto base

8

Persistência do carrinho de compras

Prof. André Nascimento Maia
Prof. Caio Nascimento Maia

Resumo

APIs REST são utilizadas tanto para leitura quanto para escrita de dados e execução de processos complexos. Neste texto, é criada uma API chamada Cart para manipulação e consulta de carrinhos de compras de um e-commerce fictício. Também é demonstrado como um componente React pode utilizar essa API.

8.1. Introdução

Web applications utilizam APIs remotas principalmente para consulta de dados, mas também para escrita e execução de rotinas complexas em um escopo protegido.

Um exemplo de rotina complexa seria um carrinho de compras em um contexto fictício de e-commerce. Um carrinho de compras com todos os produtos e configurações do usuário é uma informação importante e deve ser armazenada de forma eficiente e durável para garantir eficiência para o negócio em análises futuras e eventuais promoção enviadas ao usuários, mas também para garantir a experiência adequada para o cliente, que pode sair da página e voltar a qualquer momento para dar continuidade as suas compras com o carrinho já preenchido.

Este texto, demonstra os principais conceitos para a criação de uma API REST chamada Cart, que permite consultar um carrinho de compras de um e-commerce fictício e também atualizar as informações de um carrinho já existente.

8.2. API Cart

Para permitir a consulta e a busca de um carrinho de compras, precisamos primeiramente definir o modelo de dados que armazenamos para cada carrinho. O Código 8.1 mostra um modelo de dados implementado em Python e utilizando o sistema de mapeamento objeto-relacional SQLAlchemy simples para a persistência em uma tabela de um banco de dados SQLite em memória. Nesse exemplo, consideramos que a

lista de produtos é apenas uma lista fixa em JSON que não precisa ser normalizada seguindo as regras de normalização de bancos de dados relacionais. Essa decisão trará eficiência na implementação e simplicidade para o modelo de dados.

Código 8.1: Modelo de dados para Cart implementado com SQLAlchemy

```
01. class Cart(db.Model):
02.     """Cria a entidade de Cart"""
03.     id = db.Column(db.Integer, primary_key=True)
04.     code = db.Column(db.String(255), unique=True, nullable=False)
05.     content = db.Column(db.Text, unique=False, nullable=False)
06.     created_at = db.Column(db.DateTime, unique=False, nullable=False)
07.     updated_at = db.Column(db.DateTime, unique=False, nullable=False)
```

Fonte: autores, 2022.

O atributo *content* do Cart é a lista de produtos em JSON. O Código 8.2 mostra a lista de produtos a partir da linha 05.

Código 8.2: Modelo de resultado da consulta a API Cart

```
01.     {
02.       "code": "fixed-cart-code",
03.       "createdAt": "2022-02-07T03:16:52.801854",
04.       "id": 1,
05.       "products": [
06.         {
07.           "code": "1",
08.           "qty": 1,
09.           "title": "Caneca Personalizada de Porcelana",
10.           "tumbnailUrl": "http://meu-ecommerce.com/....jpg",
11.           "unitPrice": 123.45
12.         },
13.         {
14.           "code": "2",
15.           "qty": 2,
16.           "title": "Caneca Importada Personalizada em Diversas Cores",
17.           "tumbnailUrl": "http://meu-ecommerce.com/....jpg",
18.           "unitPrice": 123.45
19.         },
20.         {
21.           "code": "3",
22.           "qty": 1,
23.           "title": "Caneca de Tulipa",
24.           "tumbnailUrl": "http://meu-ecommerce.com/....jpg",
25.           "unitPrice": 123.45
26.         }
```

```
27.       ],

28.       "updatedAt": "2022-02-07T03:16:52.801854"
29.     }
```

Fonte: autores, 2022.

Com o modelo de dados pronto, é necessário criar as funções para manipular cada operação da API Cart. O padrão utilizado é REST (Red Hat), então é necessário mapear as ações em verbos HTTP.

Os principais verbos HTTP utilizados em APIs REST são:

- GET - Usado para obter dados de um recurso específico ou listar todos os recursos;
- POST - Utilizado para criar um dado novo para um recurso específico;
- PUT - Utilizado para criar um dado novo para um recurso específico ou sobrescrever (atualizando) um existente;
- PATCH - Utilizado para atualizar parte de um recurso já existente;
- DELETE - Utilizado para apagar os dados de um recurso específico.

A API Cart permitirá a consulta de carrinhos, retornando todos os carrinhos quando um código não é informado ou um específico quando o código é informado, e também, a criação ou sobrescrita de um carrinho caso ele já exista.

O Código 8.3 implementa a função para consulta de carrinhos e o Código 8.4 implementa a função para criação ou sobrescrita de um carrinho já existente.

Código 8.3: Endpoint para consulta da API Cart

```
01. @app.route("/carts/", methods=["GET"])
02. @app.route("/carts/<cart_code>", methods=["GET"])
03. def get(cart_code=None):
04.     """Lista todos ou exibe os detalhes de um cart quando cart_code informado."""
05.     if cart_code is not None:
06.         cart = Cart.query.filter(
07.             Cart.code == cart_code
08.         ).one()
09.         return jsonify(cart.serialized)
10.     else:
11.         carts = Cart.query.all()
12.         return jsonify({
13.             'carts': [c.serialized for c in carts]
14.         })
```

Fonte: autores, 2022.

As linhas 01 e 02 do Código 8.3 mapearam as rotas para o recurso */carts* informando explicitamente que o verbo HTTP aceito é o GET apenas. Esse é o endpoint de consulta. Quando *cart_code* é informado, faz uma consulta específica retornando o *cart* encontrado pelo código, caso contrário, lista todos os *carts* armazenados.

Código 8.4: Endpoint para criação ou sobrescrita da API Cart

```
01. @app.route("/carts/<cart_code>", methods=['PUT'])
02. def put(cart_code=None):
03.     """Atualiza um cart a partir da lista de produtos enviada no payload."""
04.     payload = request.get_json()
05.
06.     Cart.query.filter(Cart.code == cart_code).update(
07.         {
08.             Cart.content: json.dumps(payload),
09.             Cart.updated_at: datetime.utcnow()
10.         }, synchronize_session=False
11.     )
12.     db.session.commit()
13.
14.     return jsonify(Cart.query.filter(Cart.code == cart_code).one().serialized)
```

Fonte: autores, 2022.

O Código 8.4 também cria a sua rota mapeando explicitamente o verbo HTTP PUT. Esse é um *endpoint* para da API Cart para criar um novo recurso ou sobrescrevê-lo caso ele já exista.

Um ponto importante sobre o estilo arquitetural REST é que os verbos HTTP para modificação podem ser confusos de início. É um exemplo o POST e o PUT. Ambos são utilizados para criar novos dados, porém, quando utilizamos o POST, geralmente não temos uma chave ou código para referenciar um recurso específico, dessa forma uma chamada para criar um novo carrinho utilizaria a URL */carts* apenas. No nosso caso sempre utilizaremos uma chave ou código específico para identificar o recurso, o *cart_code*. Sendo assim, o verbo correto para se utilizar é o PUT, mesmo que seja a criação de um novo carrinho de compras.

Ambos os Códigos 8.3 e 8.4 fazem uso do modelo Cart do Código 8.1, o que facilita o acesso direto ao banco de dados para consulta e manipulação dos dados.

8.3. Utilizando a API Cart

Uma página para permitir a visualização e manipulação dos dados de um carrinho de compras de um e-commerce fictício é proposto na Figura 8.1. A partir desse protótipo, é proposto no Código 8.5 um componente React para o carrinho de compras.

Figura 8.1: Protótipo de carrinho de compras

Fonte: autores, 2022.

Seguindo as recomendações para criação de componentes React, a decomposição do protótipo em componentes e suas hierarquias deve ser conforme a seguir:

- Cart
  - ItemsList
    - Item
      - ItemQuantity
  - Summary

O componente Cart tem um componente mais interno chamado ItemQuantity que exibe a quantidade de produtos selecionada pelo usuário para cada item do carrinho e quando essa quantidade é alterada, um evento chamado onQuantityChanged é disparado com um evento que contém o produto alterado e a nova quantidade para o determinado produto.

O Código 8.5 é a implementação do componente Cart.

```
01. import { useEffect, useState } from "react";
02.
03. import ItemsList from "./ItemsList";
04. import Summary from "./Summary";
05. import { getCart, updateCart } from "../../api/api";
06.
07. const FIXED_CART_CODE = "fixed-cart-code";
08.
09. function Cart(props) {
10.   const [products, setProducts] = useState([]);
11.
12.   useEffect(() => {
13.     getCart(FIXED_CART_CODE).then((json) => {
14.       setProducts(json.products);
15.     });
16.   }, []);
17.
18.   function onQuantityChanged(ev) {
19.     for (const product of products) {
20.       if (product.code === ev.product.code) {
21.         product.qty = ev.newQty;
22.         setProducts(products.map((x) => x));
23.         updateCart(FIXED_CART_CODE, products);
24.         break;
25.       }
26.     }
27.   }
28.
29.   return (
30.     <div>
31.       <hr />
32.       <ItemsList products={products} onQuantityChanged={onQuantityChanged}
/>
33.       <hr />
34.       <Summary products={products} />
35.     </div>
36.   );
37. }
38.
39. export default Cart;
```

Fonte: autores, 2022.

O Código 8.5 mostra apenas o componente Cart e omite os componentes mais internos para facilitar a leitura. Através do hook useEffect, assim que o componente é criado, a função getCart com o argumento de um código fixo para um carrinho de compras é informada. Essa função, criada no Código 8.6, é uma função para consultar um carrinho específico na API REST criada para Cart.

Código 8.6: Consulta de carrinho de compras utilizando API Cart

```
01. export async function getCart(code) {
02.   return fetch(`http://127.0.0.1:5000/carts/${code}`).then((response) =>
03.     response.json()
04.   );
05. }
```

Fonte: autores, 2022.

Em seguida, toda vez que a quantidade de produtos é atualizada no carrinho pelo usuário, o evento onQuantityChanged é disparado e a função da linha 18 é invocada. Nesta função, sempre que um produto atualiza a sua quantidade na lista de produtos que é um *state* do componente Cart a função updateCart da linha 23 também é invocada. Essa função, criada no Código 8.7, é a função para criação ou sobrescrita de um carrinho de compras da API Cart.

Código 8.7: Criação ou sobrescrita de um carrinho de compras utilizando a API Cart

```
00. export async function updateCart(code, products) {
00.  return axios.put(`http://127.0.0.1:5000/carts/${code}`, products);
00. }
```

Fonte: autores, 2022.

A função updateCart do Código 8.7, recebe o código do carrinho de compras e a lista de produtos atualizada pelo usuário. Nesta função, utiliza o HTTP-Client Axios (Axios HTTP-Client, 2022) para executar um HTTP PUT na API Cart.

8.4. Conclusões

Utilizando chamadas remotas de dentro de componentes React, conseguimos integrar o *front-end* com o *back-end* de forma simples e clara utilizando APIs REST.

Foi demonstrado como executar chamadas tanto para consulta de informações quando para criação ou atualização de um carrinho de compras para um e-commerce fictício em componentes React através de uma API REST chamada API Cart que manipula e consulta os dados dos carrinhos em um banco de dados SQLite em memória.

Referências

AXIOS HTTP-CLIENT. **Getting started**. Axios. 2022. Disponível em: <https://axios-http.com/docs/intro>. Acesso em: 07 fev. 2022.

RED HAT. **O que é API? O que é API REST?** Red Hat, 8 mai. 2020. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>. Acesso em: 07 fev. 2022.