Bruno Castro Tomaz (10389988)
Gustavo Saad Maluhy Andrade (10322747)
Luiz Gabriel Profirio Mendes (10382703)
Rafaela Perrotti Zyngier (10395424)

Proposta

Desenvolver um conversor de moedas simples, focado em uma interface amigável e responsiva. O usuário poderá digitar um valor em reais (BRL) e visualizar a conversão instantânea para diferentes moedas, como USD, EUR, BTC e outras. O objetivo é criar uma aplicação informativa e fácil de usar, ideal para demonstrar conceitos de frontend com Next.js e manipulação básica de dados.

O foco deste tutorial está na explicação da implementação do código, o foco não está na explicação da utilização ou implementação dos estilos.

Página inicial, código principal

```
"use client"

import Input from "./components/Input/Input";
import CardMoeda from "./components/CardMoeda/CardMoeda";
import Grid from "./components/Grid/Grid";
import Label from "./components/Label/Label";
import { useState, useEffect } from "react";
import Container from "./components/Container/Container";

export default function Home() {
  const API_KEY = process.env.NEXT_PUBLIC_EXCHANGERATE_API_KEY;
  const BASE_URL = `https://v6.exchangerate-
api.com/v6/${API_KEY}/latest/BRL`;


  const [valorBRL, setValor] = useState(0);
  const [exchangeRates, setExchangeRates] = useState(null);
  const [loading, setLoading] = useState(true); // Adiciona um estado
de carregamento
  const [error, setError] = useState(null); // Adiciona um estado para
erros


  async function buscarTaxasDeCambio() {
    setLoading(true); // Começa a carregar
    setError(null); // Limpa erros anteriores
```

```
    if (!API_KEY) {
      console.error("Erro: A chave da API da ExchangeRate-API não foi
configurada. Verifique seu arquivo .env.local.");
      setError("Erro de configuração da API.");
      setLoading(false);
      return;
    }

    try {
      const resp = await fetch(BASE_URL);
      const json = await resp.json();

      if (json.result === "success") {
        setExchangeRates(json.conversion_rates);
        console.log(json);
      } else {
        console.error("Erro ao buscar taxas de câmbio:", json.result,
json["error-type"]);
        setError(`Erro da API: ${json["error-type"]}`);
      }
    } catch (err) {
      console.error("Erro na requisição da API de Câmbio:", err);
      setError("Erro ao buscar dados. Verifique sua conexão.");
    } finally {
      setLoading(false); //termina de carregar, independente do
resultado
    }
  }

  useEffect(() => {
    buscarTaxasDeCambio();
  }, []); // <--- CORRIGIDO: Adicione o array de dependências vazio
  //renderização condicional
  if (loading) {
    return (
      <Container>
        <p>Carregando taxas de câmbio...</p>
      </Container>
      );
  }
  if (error) {
      return (
        <Container>
          <p>Ocorreu um erro: {error}</p>
          <button onClick={buscarTaxasDeCambio}>Tentar
Novamente</button>
        </Container>
      );
  }
```

```
  // Se chegou até aqui, exchangeRates não é null e os dados foram
carregados

  const cotacaoEUR = exchangeRates.EUR;
  const cotacaoVES = exchangeRates.VES;
  const cotacaoUSD = exchangeRates.USD;
  const cotacaoARS = exchangeRates.ARS;



  //calculo inicial
  const eurRaw = valorBRL * cotacaoEUR;
  const vesRaw = valorBRL * cotacaoVES;
  const usdRaw = valorBRL * cotacaoUSD;
  const arsRaw = valorBRL * cotacaoARS;



  //arredondamento para duas casas decimais
  const eur = parseFloat(eurRaw.toFixed(2));
  const ves = parseFloat(vesRaw.toFixed(2));
  const usd = parseFloat(usdRaw.toFixed(2));
  const ars = parseFloat(arsRaw.toFixed(2));

  //console.log("Valor atual:", valorBRL);

  return (
    <Container>
      <Input onValorChange={(novoValor) => setValor(novoValor)}/>
      <Label text={"Converter para"}></Label>
      <Grid>
        <CardMoeda nome={"EUR"} valorMoeda={eur} cotacao=
{cotacaoEUR}/>
        <CardMoeda nome={"VES"} valorMoeda={ves} cotacao=
{cotacaoVES}/>
        <CardMoeda nome={"USD"} valorMoeda={usd} cotacao=
{cotacaoUSD}/>
        <CardMoeda nome={"ARS"} valorMoeda={ars} cotacao=
{cotacaoARS}/>
      </Grid>
    </Container>
  );
}
```

Configuração e Componentes Importados

- Ativa o modo `"use client"` para habilitar recursos do React no lado do cliente.
- Importa componentes reutilizáveis: `Input`, `CardMoeda`, `Grid`, `Label`, `Container`.

API de Taxas de Câmbio

- Usa uma API pública da ExchangeRate-API com chave de acesso (`API_KEY`), variável de ambiente.
- Define a URL base para buscar a cotação do **Real Brasileiro (BRL)** para outras moedas.

Estados do Componente (useState)

- `valorBRL`: valor em reais digitado pelo usuário.
- `exchangeRates`: objeto com as taxas de câmbio.
- `loading`: indica se os dados estão sendo carregados.
- `error`: armazena mensagens de erro da API ou conexão.

Função de Busca da API

- `buscarTaxasDeCambio()`:
  - Verifica se a chave da API está presente.
  - Faz `fetch` para obter as taxas.
  - Trata erros de requisição ou retorno inválido.
  - Armazena os dados no estado `exchangeRates`.

---

useEffect

- Executa `buscarTaxasDeCambio()` uma única vez quando o componente carrega.

Renderização Condicional

- Se `loading`: exibe mensagem "Carregando taxas de câmbio...".
- Se `error`: exibe mensagem de erro e botão para tentar novamente.
- Se sucesso: prossegue para exibir conversões.

Conversão de Moedas

- Extrai cotações de: EUR, VES (bolívar), USD, ARS.
- Calcula o valor convertido com base em `valorBRL`.
- Arredonda cada valor para 2 casas decimais.

Renderização da Interface

- Componente `Input`: permite ao usuário inserir um valor em reais.
- Componente `Label`: exibe o texto "Converter para".
- Componente `Grid`: organiza os `CardMoeda` de:
  - Euro (EUR)
  - Bolívar Venezuelano (VES)
  - Dólar Americano (USD)
  - Peso Argentino (ARS)

Página da Moeda

A página é carregada após clicar no elemento CardMoeda da página inicial.

```
"use client"

import { BackButton } from "@/app/components/Button/page";
import {LargeCard} from "@/app/components/CardMoeda/CardMoeda";
import Container from "@/app/components/Container/Container";
import Grid from "@/app/components/Grid/Grid";
import Input from "@/app/components/Input/Input";
import { GreenLabel } from "@/app/components/Label/Label";
import { useParams, useSearchParams } from "next/navigation";
import { useState } from "react";

export default function Moeda (){

    const [inputValor, setValor] = useState(0);

    const params = useParams();
    const nome = params.nome;
    const valorMoeda = useSearchParams().get('valorMoeda');
    const cotacao = useSearchParams().get('cotacao');
```

```
    /* Substituir os valores em parênteses pelos valores da API */
    const variacaoPercentual = (0.8).toLocaleString('pt-BR');
    const variacaoHoje = (4444.02).toLocaleString('pt-BR');
    const ultimoFechamento = (530128.76).toLocaleString('pt-BR');

    return (
        <Container>
            <Input valor={inputValor ? inputValor :
parseFloat((valorMoeda / cotacao).toFixed(2))} onValorChange=
{(novoValor) => setValor(novoValor)}></Input>
            <LargeCard nome={nome} valorMoeda={inputValor ? inputValor
* cotacao : valorMoeda}></LargeCard>
            <Grid>
                <GreenLabel text={`${variacaoPercentual}%`}>
</GreenLabel>
                <GreenLabel text={`${variacaoHoje} Hoje`}>
</GreenLabel>
            </Grid>
            <GreenLabel text={`Último fechamento
${ultimoFechamento}`}></GreenLabel>
            <BackButton text={"test"}></BackButton>
        </Container>
    );
}
```

- A página é uma **visualização detalhada da moeda** clicada na tela principal.
- Lê dados básicos da URL (nome, valor convertido, cotação).
- Permite que o usuário simule outros valores.
- Mostra valores estáticos que futuramente poderiam ser substituídos por dados reais da API (como variação e fechamento).
- `"use client"`: indica que o componente deve ser renderizado no lado do cliente, permitindo uso de hooks como `useState` e `useParams`.

Importações

- **Componentes reutilizáveis**:
  - `BackButton`: botão que leva de volta à página inicial.
  - `LargeCard`: versão maior e sem navegação dos cartões de moeda.
  - `Container`, `Grid`, `Input`, `GreenLabel`: componentes visuais.
- **Hooks de navegação** (Next.js):
  - `useParams`: extrai parâmetros dinâmicos da URL (`[nome]`).
  - `useSearchParams`: extrai parâmetros da **query string**.

Estados

- `inputValor` (React `useState`): armazena o valor numérico digitado no input, inicialmente `0`.

Parâmetros extraídos da URL

- `params.nome`: obtido a partir do caminho dinâmico da URL, como em `/moeda/USD`.
- `valorMoeda`: valor convertido da moeda, passado como query param.
- `cotacao`: cotação da moeda em relação ao BRL, também passado por query param.

Renderização

1. **Container** envolve todo o conteúdo visual.
2. **Input**:
   - Exibe um valor inicial com base na fórmula: `valorMoeda / cotacao`, convertendo de volta para BRL.
   - Permite atualização via `onValorChange`, salvando no estado `inputValor`.
3. **LargeCard**:
   - Mostra o valor convertido da moeda com base em `inputValor * cotacao`, ou usa `valorMoeda` se o input estiver vazio.
4. **Grid com dois `GreenLabel`**:
   - Exibe a variação percentual e o valor de variação do dia.
5. **Outro `GreenLabel`**:
   - Mostra o último fechamento da moeda.
6. **BackButton**:
   - Botão que redireciona de volta para a página inicial (`/`).
   - O prop `text={"test"}` não está sendo usado dentro do componente, então é inefetivo neste estado.

Componentes

Cada componente está localizado no diretório "src/app/components", cada componente tem um diretório javascript e estilo próprio. Isso é feito para se ter o desenvolvimento da maneira mais modular possível.

Nem todos os componentes serão explicados, há componentes que possuem funcionalidade puramente destinada a estilização e que não foram inclusos neste tutorial:

- Footer
- Container
- Grid
- Label
- Title

Os componentes a seguir são cruciais na funcionalidade do site:

CardMoeda

```
import styles from './cardMoeda.module.css'
import { useRouter } from 'next/navigation'

export default function CardMoeda({nome, valorMoeda, cotacao}) {
    const router = useRouter();

    return (
        <article className={styles.card} onClick={() =>
router.push(`/moeda/${nome}?
valorMoeda=${valorMoeda}&cotacao=${cotacao}`)}>
            <p id={styles.nome}>{nome}</p>
            <p id={styles.valorMoeda}>{valorMoeda}</p>
        </article>
    );
}

export function LargeCard({nome, valorMoeda}) {
    return (
        <article className={styles.largeCard} >
            <p id={styles.nome}>{nome}</p>
            <p id={styles.valorMoeda}>{valorMoeda}</p>
        </article>
    );
}
```

Importações

- `styles` — Importa os estilos CSS específicos do componente (`cardMoeda.module.css`).
- `useRouter` — Hook do Next.js para navegação entre páginas.

Componente `CardMoeda`

Esse é o **componente principal exportado por padrão**.

Props recebidas:

- `nome` — Código da moeda (ex: `"USD"`, `"EUR"`).
- `valorMoeda` — Valor convertido da moeda.
- `cotacao` — Cotação da moeda em relação ao real.

Funcionalidade:

- Usa o `useRouter` para obter o objeto de roteamento.
- Retorna um `article` estilizado com uma classe CSS (`styles.card`).
- Ao clicar no card:
  - Navega para a rota `/moeda/{nome}`.
  - Passa os dados `valorMoeda` e `cotacao` como parâmetros na query string da URL.
  - Exemplo: clicar no card de dólar leva para `/moeda/USD?valorMoeda=5.43&cotacao=5.43`.

Estrutura interna:

- Um parágrafo exibe o nome da moeda (`<p id={styles.nome}>`)
- Outro parágrafo exibe o valor convertido (`<p id={styles.valorMoeda}>`).

Componente `LargeCard`

Esse é um **componente auxiliar** exportado (não padrão).

Props recebidas:

- `nome` — Nome/código da moeda.
- `valorMoeda` — Valor convertido.

Funcionalidade:

- Retorna um `article` com uma classe diferente (`styles.largeCard`).
- Exibe os mesmos dois parágrafos que `CardMoeda`, mas sem navegação.
- Pode ser usado em outra parte do app (como uma tela de detalhe) onde não é necessário redirecionamento.

Button

```
import { useRouter } from 'next/navigation';
import styles from './button.module.css'

export function BackButton () {

    const router = useRouter();

    return (
        <button className={styles.backButton} onClick={() =>
router.push('/')}>
            <img className={styles.img} src="/back.png"></img>
        </button>
    );
}
```

Importações

- `useRouter` — Hook de navegação do Next.js, usado para redirecionar o usuário.
- `styles` — Importa os estilos definidos no arquivo CSS modular `button.module.css`.

Componente `BackButton`

Funcionalidade:

- Cria um botão estilizado com `styles.backButton`.

- Dentro do botão, há uma imagem (`/back.png`) com estilo `styles.img`, provavelmente um ícone de "voltar".
- Ao clicar no botão:
  - Usa o `router.push('/')` para redirecionar o usuário de volta à **página inicial** (`/`).

Estrutura:

- `<button>`: elemento clicável.
  - `<img>`: exibe uma imagem como conteúdo visual do botão.

Resumo:

- É um componente funcional e reutilizável para navegação.
- O botão serve como uma **interface visual para voltar à home page**.
- Usa roteamento do Next.js sem recarregar a página.

Input

```
import styles from './input.module.css'

export default function Input({ onValorChange, valor}) {

    const handleChange = (e) => {
        const valor = parseFloat(e.target.value);
        onValorChange(isNaN(valor) ? 0 : valor);
    };

    return (
        <input
            onChange={handleChange}
            className={styles.input}
            type="number"
            placeholder={valor ? `${valor} BRL` : "0,00 BRL"}
        ></input>
    );
}
```

Importações

- `styles` — Importa os estilos definidos no arquivo CSS modular `input.module.css`.

Componente `Input`

Props recebidas:

- `onValorChange` — Função callback fornecida pelo componente pai, chamada quando o valor do input muda.
- `valor` — Valor numérico (provavelmente em BRL) usado para configurar o placeholder do input.

Lógica interna

- **`handleChange`**:
  - Função chamada sempre que o usuário digita algo no campo.
  - Converte a string de entrada para `float` com `parseFloat`.
  - Se o valor convertido não for um número (`NaN`), define `0`.
  - Passa esse valor para a função `onValorChange`.

Elemento renderizado

- Um campo de input numérico (`type="number"`).
- Estilizado com `styles.input`.
- `placeholder`: mostra o valor atual com `BRL` como sufixo, ou `"0,00 BRL"` se não houver valor.