Documentação parte 2 do projeto:

Sumário

Explicação do código em Html5:

```html
<!DOCTYPE html>
<html Lang="pt">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Èrè Moeda Digital</title>
    <link rel="stylesheet" href="styles.css">
</head>
```

Essa é a estrutura inicial do nosso site, possuí linguagem em português, título, escala e está diretamente linkado com a nossa folha de estilo css.

```html
<body>
    <header>
        <img src="logo.png" alt="Logo Èrè" class="logo">
        <div class="header-content">
            <h1>Èrè Moeda Digital</h1>
            <p>A melhor moeda da atualidade!</p>
            <a href="#compreAqui" class="btn-destaque">Comece Agora</a>
        </div>
    </header>
```

Para darmos início a folha criamos uma tag chamada body e dentro dela criamos outra chamada header que é onde está localizada a nossa logo e o título do nosso site.

```
<main>
    <div class="container">
        <div class="coluna-esquerda">
            <div class="menu-lateral">
                <h3>Menu</h3>
                <ul>
                    <li><a href="#quemSomos">Quem Somos</a></li>
                    <li><a href="#nossaVisao">Nossa Visão</a></li>
                    <li><a href="#parceiros">Parceiros</a></li>
                    <li><a href="#quemAjudamos">Quem Ajudamos</a></li>
                    <li><a href="#compreAqui">Compre Moedas</a></li>
                </ul>
            </div>
        </div>
```

Em seguida criamos uma tag main e dentro dela criamos alguns containers: Um container principal a coluna de navegação um menu e para o menu criamos uma lista não ordenada com itens de navegação que são links.

```
<div class="coluna-direita">
    <section id="inicio">
        <h2>Bem-vindo ao Èrè Moeda Digital</h2>
        <p>Selecione uma opção no menu ao lado para ver mais informações.</p>
    </section>

    <section id="quemSomos" class="conteudo-oculto">
        <h2>Quem Somos</h2>
        <p>Somos uma plataforma inovadora que oferece a moeda digital mais recompensadora do mercado.</p>
    </section>

    <section id="nossaVisao">
        <h2>Nossa Visão</h2>
        <p>Queremos transformar a forma como as pessoas interagem com moedas digitais, oferecendo benefícios para consumidores
    </section>

    <section id="parceiros">
        <h2>Nossos Parceiros</h2>
        <p>Empresar parceiras que nos ajudam a tornar esse projeto uma realidade.</p>
        <div class="grid-container">
            <img src="./parceiros_1.jpeg" alt="Parceiro 1">
            <img src="./parceiros_2.jpeg" alt="Parceiro 2">
            <img src="./parceiros_3.jpeg" alt="Parceiro 3">
            <img src="./parceiros_4.jpeg" alt="Parceiro 4">
        </div>
    </section>
```

Nesta parte do código criamos mais um container para abrigar seções com explicações sobre o nosso produto. E uma pequena explicação sobre os nossos parceiros com imagens das empresas.

```
<section id="quemAjudamos">
    <h2>Quem Ajudamos</h2>
    <p>Além de garantir produtos com preços acessíveis, você ajuda comunidades necessitadas na África.</p>
    <div class="grid-container">
        <img src="./necessitados_1.png" alt="Necessitado 1">
        <img src="./necessitados_2.png" alt="Necessitado 2">
        <img src="./necessitados_3.png" alt="Necessitado 3">
        <img src="./necessitados_4.png" alt="Necessitado 4">
    </div>
</section>
```

Nesta sessão abaixo da anterior, possui uma explicação de quem são as pessoas que ajudarão comprando a moeda erè com imagens detalhadas.

```
<section id="compreAqui">
    <h2>Compre Moedas</h2>
    <div class="formContainer">
        <div class="quantidadeMoedas">
            <label for="quantidade">Quantidade de Moedas:</label>
            <input type="number" id="quantidade" name="quantidade" min="1" value="1">
            <p>Total: <span id="total">R$ 10,00</span></p>
        </div>
        <div class="dadosPagamento">
            <form>
                <label for="nome">Nome no Cartão:</label>
                <input type="text" id="nome" name="nome" required>

                <label for="numero">Número do Cartão:</label>
                <input type="text" id="numero" name="numero" required>

                <label for="validade">Validade:</label>
                <input type="text" id="validade" name="validade" placeholder="MM/AA" required>

                <label for="cvv">CVV:</label>
                <input type="text" id="cvv" name="cvv" required>

                <button type="submit">Finalizar Compra</button>
            </form>
            <div class="cartaoImagem">
                <img src="cartao.png" alt="Cartão de Crédito">
```

Nesta parte criamos mais uma sessão para o usuário poder acrescentar os seus dados bancários e comprar a moeda digital erè. Fizemos isso criando alguns inputs, um botão para o envio das informações e uma imagem que futuramente acrescentaremos Javascript para aparecer dinamicamente a bandeira do cartão para passar credibilidade ao cliente.

```
<footer>
    <p>Todos os direitos reservados ©</p>
    <p>Produzido por Eduardo Pirolo & Marco Aurélio</p>
</footer>
```

Criamos por ultimo uma tag footer para abordar o nome dos integrantes do site e os direitos sobre o produto.

Explicação sobre o código em Css:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

Inicialmente acrescentamos as configurações iniciais para deixar tudo o mais cru possível para podermos estilizar a pagina.

```css
body {
    font-family: 'Arial', sans-serif;
    line-height: 1.6;
    background-color: #f4f4f4;
    color: #09122C;
}
```

Aqui configuramos o body, a fonte, cor de fundo e cor das letras.

```css
header {
    background: url('background.jpg') no-repeat center center/cover;
    color: #fff;
    padding: 60px 20px;
    text-align: center;
    position: relative;
    overflow: hidden;
}
```

No header acrescentamos uma imagem e estilizamos ela conforme o wireframe.

```css
header::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(9, 18, 44, 0.7);
    z-index: 1;
}

header .header-content {
    position: relative;
    z-index: 2;
}
```

O <header> terá um fundo escuro semitransparente cobrindo toda sua área, enquanto os elementos dentro de .header-content aparecerão acima desse fundo.

```css
header h1 {
    font-size: 3em;
    margin-bottom: 10px;
    animation: fadeIn 2s ease-in-out;
}

header p {
    font-size: 1.5em;
    margin-bottom: 20px;
    animation: fadeIn 2.5s ease-in-out;
}

header .Logo {
    position: absolute;
    top: 20px;
    right: 20px;
    width: 100px;
    height: auto;
    z-index: 3;
}
```

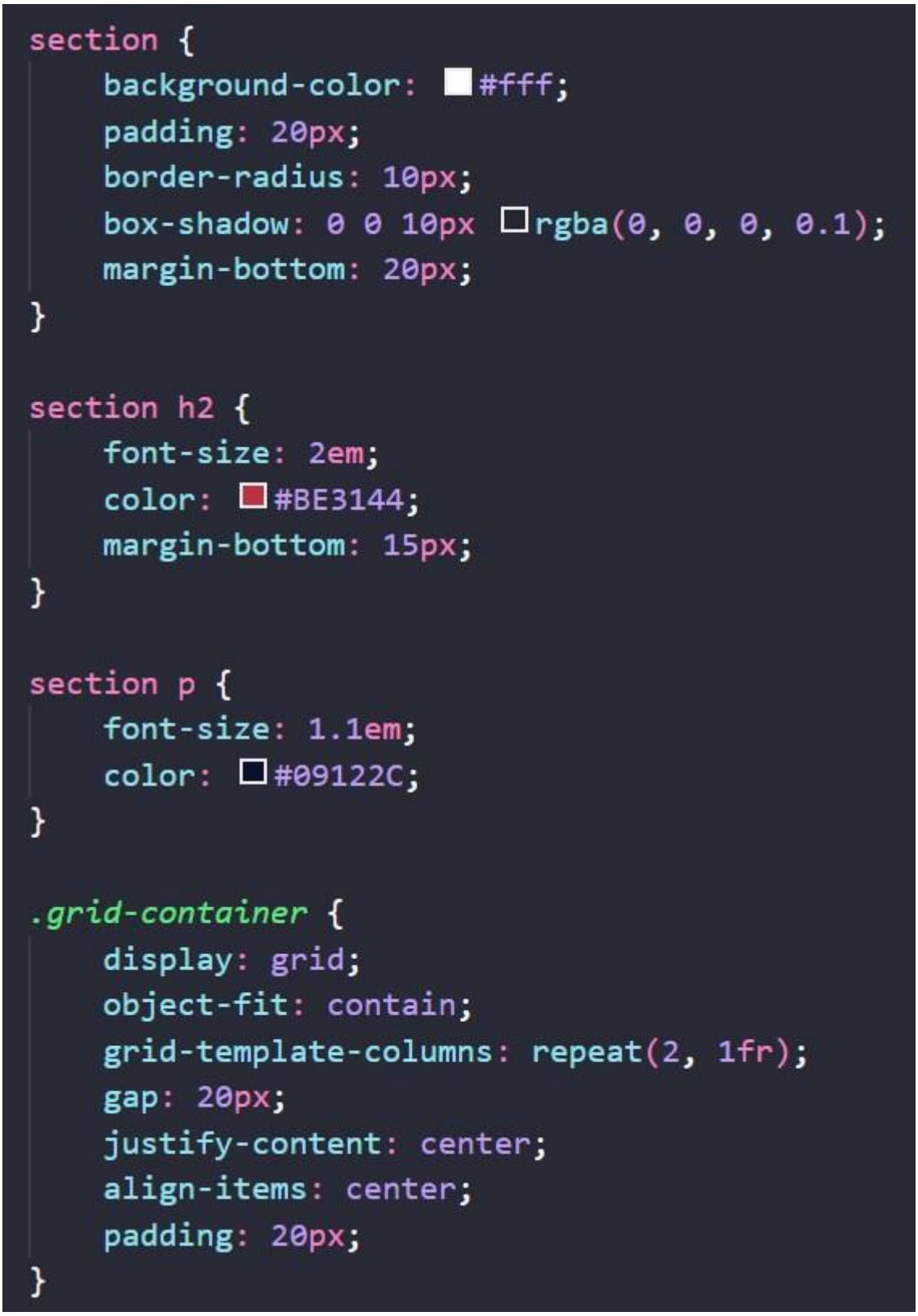
estiliza o <header>, aplicando animações suaves ao título (h1, 2s) e ao parágrafo (p, 2.5s). O logotipo (.Logo) é posicionado no canto superior direito (absolute) com z-index: 3 para ficar acima de outros elementos.

```css
.btn-destaque {
    display: inline-block;
    padding: 10px 20px;
    font-size: 1.2em;
    background-color: #BE3144;
    color: #fff;
    text-decoration: none;
    border-radius: 5px;
    transition: background-color 0.3s ease, transform 0.3s ease;
}

.btn-destaque:hover {
    background-color: #872341;
    transform: scale(1.1);
}

.container {
    display: flex;
    gap: 20px;
    max-width: 1200px;
    margin: 20px auto;
    padding: 0 20px;
}

.coluna-esquerda {
    flex: 1;
}
```

Este código CSS estiliza um botão (.btn-destaque) com cor de fundo vermelha (#BE3144), texto branco, bordas arredondadas e efeito de transição suave ao passar o mouse, mudando a cor para um tom mais escuro (#872341) e aumentando ligeiramente o tamanho (scale(1.1)). Além disso, define um layout flexível para .container, com espaçamento (gap: 20px), largura máxima de 1200px, centralização automática e preenchimento lateral. A classe .coluna-esquerda recebe flex: 1, permitindo que ela ocupe espaço proporcional dentro do container flexível.

```css
.coluna-direita {
    flex: 3;
}

.menu-lateral {
    background-color: #fff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.menu-lateral h3 {
    font-size: 1.5em;
    color: #BE3144;
    margin-bottom: 15px;
}

.menu-lateral ul {
    list-style: none;
}

.menu-lateral ul li {
    margin-bottom: 10px;
}
```

Define estilos para duas classes principais: .coluna-directa e .menu-Lateral. A classe .coluna-directa tem uma propriedade flex definida como 3, o que significa que ela ocupará três vezes mais espaço em um layout flexível em comparação com outros elementos flexíveis. A classe .menu-Lateral estiliza um menu lateral com um fundo branco (#ffff), padding de 20px, bordas arredondadas e uma sombra suave. O título dentro do menu (h3) tem um tamanho de fonte de 1.5em e uma cor vermelha (#BE3144), com uma margem inferior de 15px. A lista (ul) dentro do menu não tem marcadores (list-style: none), e cada item da lista (li) tem uma margem inferior de 10px para espaçamento entre os itens.



```css
section {
    background-color: #fff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin-bottom: 20px;
}

section h2 {
    font-size: 2em;
    color: #BE3144;
    margin-bottom: 15px;
}

section p {
    font-size: 1.1em;
    color: #09122C;
}

.grid-container {
    display: grid;
    object-fit: contain;
    grid-template-columns: repeat(2, 1fr);
    gap: 20px;
    justify-content: center;
    align-items: center;
    padding: 20px;
}
```

O código CSS apresentado define estilos para elementos section, h2, p e uma classe .grid-container. As seções (section) têm um fundo branco (#ffff), padding de 20px, bordas arredondadas, uma sombra suave e uma margem inferior de 20px para espaçamento. Os títulos (h2) dentro das seções têm um tamanho de fonte de 2em, cor vermelha (#BE3144) e uma margem inferior de 15px. Os parágrafos (p) têm um

tamanho de fonte de 1.1em e cor azul escuro (#09122C). A classe .gridcontainer cria um layout de grade com duas colunas de largura igual (repeat(2, 1fr)), um espaçamento (gap) de 20px entre os itens, e centraliza o conteúdo tanto horizontal quanto verticalmente. O padding de 20px adiciona espaço interno ao contêiner da grade.

```css
.grid-container img {
    width: 250px;
    height: 180px;
    object-fit: contain;
    border-radius: 15px;
    padding: 10px;
    background-color: #fff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.grid-container img:hover {
    transform: scale(1.05);
    box-shadow: 0 0 15px rgba(135, 35, 65, 0.3);
}

.cartaoImagem {
    text-align: center;
    margin-top: 20px;
}

.cartaoImagem img {
    width: 200px;
    height: auto;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
}
```

O código define estilos para imagens dentro de um contêiner de grade e para uma classe .cartaoImagem. As imagens dentro do .grid-container têm uma largura fixa de 250px e altura de 180px, com object-fit: contain para garantir que a imagem se ajuste sem distorção. Elas têm bordas arredondadas, padding interno, fundo branco e uma sombra suave. Há também uma transição suave para efeitos de escala e sombra ao passar o mouse sobre as imagens, aumentando ligeiramente o tamanho e a intensidade da sombra. A classe .cartaoImagem centraliza o texto e adiciona uma margem superior de 20px. As imagens dentro desta classe têm uma largura fixa de

200px, altura automática, bordas arredondadas e uma sombra mais pronunciada. Esses estilos visam melhorar a apresentação visual e a interatividade das imagens.

```css
.formContainer {
    display: flex;
    gap: 20px;
}

.quantidadeMoedas, .dadosPagamento {
    flex: 1;
    background-color: #f9f9f9;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.quantidadeMoedas label, .dadosPagamento label {
    display: block;
    margin-bottom: 10px;
    font-weight: bold;
    color: #09122C;
}

.quantidadeMoedas input, .dadosPagamento input {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
```

O código define estilos para um contêiner de formulário e seus elementos internos. O _formContainer utiliza um layout flexível com um espaçamento (gap) de 20px entre os itens. As classes _quantidadeMoedas_ e _dadosPagamento têm fundo cinza claro (#f9f9f9), padding de 20px, bordas arredondadas e uma sombra suave. Os rótulos (label) dentro dessas classes são exibidos em bloco, com margem inferior de 10px, texto em negrito e cor azul escuro (#09122C). Os campos de entrada (input) ocupam 100% da largura, têm padding de 10px, margem inferior de 15px, borda cinza (#ccc) e bordas arredondadas. Esses estilos visam criar um formulário organizado e visualmente agradável.

```css
.dadosPagamento button {
    background-color: #BE3144;
    color: #fff;
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.dadosPagamento button:hover {
    background-color: #872341;
}

footer {
    text-align: center;
    padding: 20px;
    background-color: #09122C;
    color: #fff;
    margin-top: 20px;
}

footer p {
    margin: 5px 0;
}

@media (max-width: 768px) {
    .container {
        flex-direction: column;
    }
```

O código define estilos para um botão de pagamento, um rodapé e uma regra de mídia para telas menores. O botão dentro de .dadosPagamento tem um fundo vermelho (#BE3144), texto branco, padding de 10px 20px, bordas arredondadas e uma transição

suave para a cor de fundo ao passar o mouse, mudando para um tom mais escuro de vermelho (#872341). O rodapé (footer) é centralizado, com padding de 20px, fundo azul escuro (#09122C), texto branco e uma margem superior de 20px. Os parágrafos (p) dentro do rodapé têm margens reduzidas. A regra de mídia ajusta o layout para telas com largura máxima de 768px, mudando a direção do .container para coluna, o que é útil para designs responsivos. Esses estilos visam melhorar a usabilidade e a aparência em diferentes dispositivos.

```css
@media (max-width: 768px) {
    .container {
        flex-direction: column;
    }

    .menu-lateral {
        margin-bottom: 20px;
    }

    .grid-container {
        grid-template-columns: repeat(1, 1fr);
    }

    .grid-container img {
        width: 100%;
        max-width: 250px;
        height: 180px;
    }
}
```

define regras de mídia para telas com largura máxima de 768px, visando melhorar a responsividade do layout. Quando a tela é menor que 768px, o .container muda sua direção para coluna (flex-direction: column), empilhando os elementos verticalmente. O .menu-Lateral recebe uma margem inferior de 20px para espaçamento. O .grid-container ajusta seu layout para uma única coluna (gridtemplate-columns: repeat(1, 1fr)), garantindo que os itens sejam exibidos um abaixo do outro. As imagens dentro do .grid-container ocupam 100% da largura, mas têm uma largura máxima de 250px e altura fixa de 180px, mantendo a proporção e a consistência visual em dispositivos menores. Essas

alterações visam garantir uma experiência de usuário mais adequada em dispositivos móveis.

```
@media (max-width: 480px) {
    .grid-container {
        grid-template-columns: repeat(1, 1fr);
    }

    .grid-container img {
        width: 100%;
        max-width: 200px;
        height: 150px;
    }

    .cartaoImagem img {
        width: 150px;
    }
}
```

O código define regras de mídia para telas com largura máxima de 480px, visando otimizar a exibição em dispositivos móveis pequenos. Quando a tela é menor que 480px, o .grid-container ajusta seu layout para uma única coluna (grid-templatecolumns: repeat(1, 1fr)), garantindo que os itens sejam exibidos um abaixo do outro. As imagens dentro do .grid-container ocupam 100% da largura, mas têm uma largura máxima de 200px e altura fixa de 150px, mantendo a proporção e a consistência visual. Além disso, as imagens dentro da classe .cartaoImagem são redimensionadas para uma largura de 150px, adaptando-se melhor ao espaço disponível em telas menores. Essas alterações visam melhorar a usabilidade e a aparência em dispositivos móveis de pequeno porte.

```css
@media (max-width: 1200px) {
    .container {
        flex-direction: column;
        padding: 0 10px;
    }

    .coluna-esquerda, .coluna-direita {
        flex: 1 1 100%;
    }

    .menu-lateral {
        margin-bottom: 20px;
    }

    .formContainer {
        flex-direction: column;
    }

    .quantidadeMoedas, .dadosPagamento {
        width: 100%;
    }

    .cartaoImagem {
        margin-top: 20px;
        text-align: center;
    }

    .cartao {
        width: 100%;
        max-width: 300px;
        margin: 0 auto;
    }
```

Este trecho de código CSS define um **media query** que aplica estilos específicos quando a largura da tela é **igual ou menor que 1200px**, adaptando o layout para dispositivos com telas menores (responsividade). Ele reorganiza o layout em colunas para uma estrutura em **formato vertical**, alterando flex-direction: column nos containers principais. Além disso, define larguras de 100% para diversos elementos (.coluna-esquerda, .coluna-direita, .quantidadeMoedas, .dadosPagamento, .cartao) para que

ocupem toda a largura disponível. Também adiciona espaçamento inferior no menu lateral e centraliza a imagem do cartão. Essas alterações tornam a interface mais amigável em dispositivos móveis ou janelas reduzidas, facilitando a leitura e navegação.

```css
@media (max-width: 768px) {
    header h1 {
        font-size: 2em;
        margin-top: 20px;
    }

    header p {
        font-size: 1.2em;
    }

    header .logo {
        width: 80px;
        top: 10px;
        right: 10px;
    }

    .btn-destaque {
        font-size: 1em;
        padding: 8px 16px;
    }

    .menu-lateral h3 {
        font-size: 1.2em;
    }

    section h2 {
        font-size: 1.5em;
    }

    section p {
        font-size: 1em;
    }

    .grid-container {
        grid-template-columns: repeat(1, 1fr);
    }

    .grid-container img {
        max-width: 100%;
        height: 200px;
    }

    .cartao {
        width: 100%;
        max-width: 250px;
    }
}
```

Esse bloco de CSS define uma **media query para telas com largura máxima de 768px**, otimizando a exibição para tablets e dispositivos móveis menores. Ele ajusta tamanhos de fonte e margens para melhor legibilidade e aproveitamento de espaço. Por exemplo,

os títulos (h1, h2) e parágrafos (p) têm o tamanho da fonte reduzido; a logo do cabeçalho é redimensionada e reposicionada; e botões como .btn-destaque ganham padding mais adequado. A estrutura do layout também se adapta: o .grid-container passa a ter apenas uma coluna, tornando o conteúdo mais linear e responsivo, e as imagens dentro dele são redimensionadas para se ajustarem à nova largura. Além disso, o .cartao tem seu tamanho limitado a 250px para manter a proporção em telas menores. Tudo isso contribui para uma melhor usabilidade e visualização em dispositivos móveis.

```css
@media (max-width: 480px) {
    header {
        padding: 40px 10px;
    }

    header h1 {
        font-size: 1.5em;
        margin-top: 30px;
    }

    header p {
        font-size: 1em;
    }

    header .Logo {
        width: 60px;
        top: 5px;
        right: 5px;
    }

    .btn-destaque {
        font-size: 0.9em;
        padding: 6px 12px;
    }

    .menu-lateral h3 {
        font-size: 1em;
    }

    section h2 {
        font-size: 1.2em;
    }

    section p {
        font-size: 0.9em;
    }

    .grid-container img {
        max-width: 100%;
        height: 200px;
    }

    .cartao {
        width: 100%;
        max-width: 200px;
    }

    .numeroCartao {
        font-size: 1em;
    }

    .nomeCartao, .validadeCartao {
        font-size: 0.8em;
    }

    .cvvCartao {
        font-size: 0.9em;
    }
}
```

Este trecho define uma **media query para telas com largura máxima de 480px**, voltada especialmente para smartphones. Ele ajusta diversos elementos para manter a legibilidade e usabilidade em telas muito pequenas. O cabeçalho (header) recebe menos padding e suas fontes são diminuídas. A logo também é redimensionada e

reposicionada. Títulos, parágrafos, botões e menus laterais têm os tamanhos de fonte reduzidos para evitar que o conteúdo fique espremido ou estoure os limites da tela. A imagem dentro do .grid-container continua ocupando 100% da largura com altura fixa, enquanto o componente .cartao tem seu max-width limitado a 200px. Além disso, elementos de input como .numeroCartao, .nomeCartao, .validadeCartao e .cvvCartao têm suas fontes ajustadas para se encaixarem melhor no espaço disponível. Em resumo, esse conjunto de estilos garante que a interface permaneça funcional, limpa e agradável em dispositivos móveis.

Explicação do código em Javascript:

Esse código JavaScript faz duas funções principais:

1. **Atualizar o valor total de moedas** com base na quantidade informada pelo usuário.
2. **Atualizar a exibição de um cartão virtual** conforme o usuário preenche os campos do formulário.

```
const quantidadeInput = document.getElementById('quantidade');
const totalElement = document.getElementById('total');
const valorUnitario = 1.50;
```

Pega a referência dos elementos HTML:

- quantidadeInput: Campo onde o usuário digita a quantidade de moedas.
- totalElement: Onde será mostrado o valor total.
- valorUnitario: Valor unitário da moeda (R$ 1,50).

```
function atualizarTotal() {
    const quantidade = quantidadeInput.value;
    const total = quantidade * valorUnitario;
    totalElement.textContent = `R$ ${total.toFixed(2)}`;
}
```

Quando o usuário altera a quantidade, essa função:

- Obtém o valor digitado.

- Calcula o total (quantidade * valorUnitario).
- Exibe o valor formatado (duas casas decimais) no elemento totalElement.

```
quantidadeInput.addEventListener('input', atualizarTotal);
atualizarTotal();
```

A função atualizarTotal() é chamada inicialmente para exibir o valor correto ao carregar a página.

O evento input detecta qualquer alteração no campo quantidadeInput e chama a função para atualizar o total.

```
const nomeInput = document.getElementById('nome');
const numeroInput = document.getElementById('numero');
const validadeInput = document.getElementById('validade');
const cvvInput = document.getElementById('cvv');

const numeroCartao = document.querySelector('.numeroCartao');
const nomeCartao = document.querySelector('.nomeCartao');
const validadeCartao = document.querySelector('.validadeCartao');
const cvvCartao = document.querySelector('.cvvCartao');
const cartao = document.querySelector('.cartao');
```

Obtém referências dos campos de entrada do formulário (nome, número, validade, cvv).

Obtém referências dos elementos onde os dados serão exibidos no cartão virtual.

```
numeroInput.addEventListener('input', () => {
    let numero = numeroInput.value.replace(/\s/g, '');
    numero = numero.replace(/(\d{4})/g, '$1 ').trim();
    numeroCartao.textContent = numero || '#### #### #### ####';
});
```

- Remove espaços do número do cartão.
- Formata o número, adicionando um espaço a cada 4 dígitos.
- Exibe o número no cartão virtual, ou um valor padrão se o campo estiver vazio.

```
// Atualiza o nome do titular
nomeInput.addEventListener('input', () => {
    nomeCartao.textContent = nomeInput.value || 'NOME DO TITULAR';
});
```

- Atualiza o nome exibido no cartão.
- Se o campo estiver vazio, exibe um placeholder (NOME DO TITULAR).

```
// Atualiza a validade
validadeInput.addEventListener('input', () => {
    validadeCartao.textContent = validadeInput.value || 'MM/AA';
});
```

Atualiza a data de validade no cartão virtual.

```
// Atualiza o CVV e vira o cartão
cvvInput.addEventListener('focus', () => {
    cartao.classList.add('flip');
});
```

- Quando o campo CVV recebe foco (focus), adiciona a classe flip ao cartão.
- Essa classe pode ser usada no CSS para virar o cartão e mostrar o CVV.

```
cvvInput.addEventListener('blur', () => {
    cartao.classList.remove('flip');
});
```

- Quando o campo CVV perde o foco (blur), remove a classe flip, voltando à exibição normal.

js

```
cvvInput.addEventListener('input', () => {
    cvvCartao.textContent = cvvInput.value || '***';
});
```

Atualiza o CVV exibido no cartão virtual.

```
// Funções de validação
function validarNumeroCartao(numero) {
    const numeroLimpo = numero.replace(/\s/g, '');
    if (!/^\d{16}$/.test(numeroLimpo)) return false;

    // Algoritmo de Luhn
    let soma = 0;
    for (let i = 0; i < 16; i++) {
        let digito = parseInt(numeroLimpo[i]);
        if (i % 2 === 0) {
            digito *= 2;
            if (digito > 9) digito -= 9;
        }
        soma += digito;
    }
    return soma % 10 === 0;
}
```

O trecho de código em JavaScript implementa uma função chamada validarNumeroCartao, que verifica se um número de cartão de crédito é válido usando o Algoritmo de Luhn. Primeiro, ele remove espaços em branco do número informado e verifica se ele contém exatamente 16 dígitos numéricos. Se não atender a esse critério, retorna false. Em seguida, aplica o Algoritmo de Luhn: percorre cada dígito do número, dobrando os dígitos em posições pares (índice par), subtraindo 9 se o resultado for maior que 9, e somando todos os dígitos. Ao final, se a soma total for divisível por 10, retorna true, indicando um número de cartão válido; caso contrário, retorna false.

```
function validarValidade(validade) {
    const [mes, ano] = validade.split('/').map(Number);
    if (!mes || !ano || mes < 1 || mes > 12) return false;

    const agora = new Date();
    const anoAtual = agora.getFullYear() % 100;
    const mesAtual = agora.getMonth() + 1;

    return (ano > anoAtual || (ano === anoAtual && mes >= mesAtual));
}

function validarCVV(cvv) {
    return /^\d{3,4}$/.test(cvv);
}
```

Este código JavaScript contém duas funções para validar informações de cartões de crédito: a validade e o CVV. A função validarValidade verifica se a data de validade fornecida no formato "MM/AA" representa um mês entre 1 e 12 e se está no futuro ou no mês atual do ano atual. Para isso, ela divide a string, converte para números e compara com a data atual, utilizando Date. Já a função validarCVV verifica se o CVV possui exatamente 3 ou 4 dígitos numéricos, utilizando uma expressão regular. Ambas as funções retornam true se os dados forem válidos, e false caso contrário.

```
// Validação ao enviar o formulário
document.querySelector('form').addEventListener('submit', (e) => {
    e.preventDefault();

    // Valida todos os campos
    const isNumeroValido = validarNumeroCartao(numeroInput.value);
    const isValidadeValida = validarValidade(validadeInput.value);
    const isCVVValido = validarCVV(cvvInput.value);
    const isNomeValido = nomeInput.value.trim().length > 0;
```

Esse trecho de código adiciona um ouvinte de evento ao formulário HTML, interceptando o envio padrão com e.preventDefault() para que seja feita a validação dos campos antes de prosseguir. Ele chama as funções de validação validarNumeroCartao, validarValidade e validarCVV para verificar, respectivamente, se o número do cartão, a validade e o CVV estão corretos. Além disso, também checa se o campo de nome (nomeInput) não está vazio após remover espaços em branco com trim(). Cada validação retorna um valor booleano indicando se o campo é válido ou não. Esses valores podem ser usados depois para exibir mensagens de erro ou impedir o envio do formulário caso alguma informação esteja incorreta.

```
// Aplica estilos de erro
numeroInput.style.borderColor = isNumeroValido ? '#4CAF50' : '#BE3144';
validadeInput.style.borderColor = isValidadeValida ? '#4CAF50' : '#BE3144';
cvvInput.style.borderColor = isCVVValido ? '#4CAF50' : '#BE3144';
nomeInput.style.borderColor = isNomeValido ? '#4CAF50' : '#BE3144';

if (isNumeroValido && isValidadeValida && isCVVValido && isNomeValido) {
    alert('Pagamento processado com sucesso!');
} else {
    alert('Por favor, corrija os erros no formulário.');
}
```

Este trecho de código aplica estilos visuais de erro aos campos de um formulário com base nos resultados das validações feitas anteriormente. Ele altera a cor da borda de cada campo (numeroInput, validadeInput, cvvInput e nomeInput), usando verde (#4CAF50) para campos válidos e vermelho (#BE3144) para os inválidos. Em seguida, verifica se todos os campos são válidos. Se forem, exibe um alert com a mensagem "Pagamento processado com sucesso!"; caso contrário, mostra um alerta pedindo ao usuário para corrigir os erros no formulário. Esse feedback visual e textual melhora a usabilidade, ajudando o usuário a identificar e corrigir rapidamente os problemas nos dados inseridos.

```
// Validação em tempo real (on blur)
numeroInput.addEventListener('blur', () => {
    const valido = validarNumeroCartao(numeroInput.value);
    numeroInput.style.borderColor = valido ? '#4CAF50' : '#BE3144';
});

validadeInput.addEventListener('blur', () => {
    const valido = validarValidade(validadeInput.value);
    validadeInput.style.borderColor = valido ? '#4CAF50' : '#BE3144';
});

cvvInput.addEventListener('blur', () => {
    const valido = validarCVV(cvvInput.value);
    cvvInput.style.borderColor = valido ? '#4CAF50' : '#BE3144';
});
```

Esse trecho de código implementa validação em tempo real dos campos do formulário utilizando o evento blur, que é disparado quando o usuário sai de um campo de input. Para cada campo (numeroInput, validadeInput e cvvInput), é registrado um addEventListener que, ao perder o foco, chama a função de validação correspondente. Dependendo se o valor é válido ou não, a cor da borda do campo é alterada para verde (#4CAF50) ou vermelho (#BE3144). Isso fornece um feedback imediato ao usuário enquanto ele preenche o formulário, ajudando a detectar e corrigir erros antes mesmo de tentar enviar os dados.