Layout

Introdução

Nesta aula, vamos explorar o conceito de layouts no desenvolvimento front-end, abordando os diferentes tipos que podemos utilizar em HTML e CSS. Um layout define como os elementos de uma página web são organizados e apresentados ao usuário, sendo crucial para a criação de interfaces eficientes e atraentes. Vamos discutir três tipos principais de layout: centralizado, fixo e elástico, destacando suas características, vantagens, desvantagens e exemplos práticos de implementação.

Tipos de layout

Hoje, vamos explorar os diferentes tipos de layout que podemos utilizar no HTML e CSS, abordando suas características e como podemos implementá-los. Existem diversos tipos de layout, cada um com suas especificidades e aplicações. Vamos discutir alguns deles, mostrando exemplos práticos em código HTML e CSS, para que você possa entender como desenvolver esses layouts e aplicar no seu projeto.

Primeiro, vamos falar sobre o layout centralizado. Este tipo de layout é bastante utilizado para criar páginas que precisam de uma estrutura bem definida e centrada na tela, independente do tamanho da janela do navegador. Um exemplo clássico seria um site de portfólio, onde o conteúdo principal precisa estar destacado e centralizado para uma melhor visualização. A implementação desse layout pode ser feita utilizando margens automáticas em CSS, alinhando os elementos ao centro da página.

Outro tipo comum é o layout fixo. Esse layout possui uma largura definida, que não muda conforme a janela do navegador é redimensionada. É muito utilizado em sites onde o conteúdo não deve se adaptar a diferentes tamanhos de tela, garantindo que a apresentação visual permaneça consistente. Por exemplo, um blog pessoal pode usar um layout fixo para manter o texto e as imagens em uma largura específica, proporcionando uma leitura confortável.

Por fim, temos o layout elástico. Esse layout ajusta sua largura com base no tamanho da fonte, tornando-se ideal para aplicações que precisam ser responsivas ao zoom do navegador ou ao tamanho do texto definido pelo usuário. Um exemplo seria uma página de leitura, onde o texto e as imagens precisam se adaptar conforme o usuário altera as configurações de acessibilidade do navegador. A implementação pode ser feita usando unidades relativas como “em” ou “rem” no CSS.

É essencial conhecer esses layouts, pois eles são a base sobre a qual colocamos nosso conteúdo em HTML e o estilizamos em CSS, tornando o site ou aplicação mais utilizável e acessível. No nosso exemplo prático, vamos demonstrar como criar um grid layout, um flexbox layout e um float layout, três dos tipos mais utilizados na web.

Vamos começar com o grid layout. Utilizando o CSS Grid, podemos criar uma estrutura de layout complexa com linhas e colunas, permitindo um controle preciso sobre a disposição dos elementos. Um exemplo prático seria um dashboard de administração, onde diferentes widgets são organizados em uma grade para fácil acesso e visualização.

Em seguida, temos o flexbox layout. Este layout é excelente para distribuir espaço entre os itens de um contêiner, alinhando-os de maneira responsiva. Um exemplo seria um menu de navegação horizontal que precisa se ajustar ao tamanho da tela, mantendo o espaçamento adequado entre os itens.

Por último, o float layout. Embora menos utilizado atualmente devido às alternativas mais modernas, o layout flutuante ainda é útil para certos tipos de design. Um exemplo clássico seria a criação de uma galeria de imagens, onde os elementos flutuam ao lado uns dos outros, quebrando a linha conforme necessário para se ajustar ao contêiner.

Layout centralizado

Nesta segunda parte, abordaremos como criar um layout centralizado utilizando HTML e CSS. A centralização de elementos é uma prática comum e essencial para a construção de interfaces web agradáveis e funcionais. Utilizaremos principalmente o CSS para realizar a estilização necessária e alcançar o efeito desejado.

Vamos começar com uma breve introdução ao conceito de layout centralizado. Veja abaixo um exemplo prático de como criar um layout centralizado utilizando HTML e CSS:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Layout Centralizado</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="CenteredContainer">
        <div class="content">
            <h1>Bem-vindo ao Layout Centralizado</h1>
            <p>Este é um exemplo de layout centralizado.</p>
        </div>
    </div>
</body>
```

CSS (styles.css):

```css
body, html {
    margin: 0;
    height: 100%;
    display: flex;
    align-items: center;
    justify-content: center;
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
}
```

```css
.CenteredContainer {
    width: 80%;
    max-width: 600px;
    background-color: white;
    padding: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}

.content h1 {
    margin: 0;
    font-size: 24px;
}
```

```css
.content p {
    margin: 10px 0 0;
    font-size: 18px;
}
```

Nesse exemplo, o contêiner ‘.CenteredContainer’ centraliza o conteúdo na tela, independentemente do tamanho da janela do navegador. Isso é útil para páginas de login ou landing pages que precisam focar o conteúdo principal no centro da tela.

Basicamente, esse layout organiza os elementos de forma que fiquem centralizados na tela, independentemente do tamanho da janela do navegador. A estrutura básica de HTML para este tipo de layout inclui a declaração do doctype, a tag HTML com o atributo de linguagem definido como português brasileiro, e a seção head, onde inserimos os metadados, o charset, o viewport para responsividade, o título da página e o link para o arquivo CSS externo.

Passando para a prática, no corpo do HTML, criamos uma div com a classe CenteredContainer. Essa div será o contêiner principal que centralizará seu conteúdo. Dentro dela, colocamos outra div com a classe content, contendo um título (h1) e um parágrafo com texto. Essa estrutura básica será estilizada no CSS para obter o layout centralizado.

No CSS, definimos estilos para o body e o HTML, removendo margens padrões e configurando a altura. Utilizamos display: flex para definir um layout do tipo flexbox, o que facilita a centralização dos elementos. Usamos as propriedades align-items e justify-content com o valor center para alinhar os itens centralmente tanto horizontal quanto verticalmente. Definimos a fonte como Arial e a cor de fundo como cinza claro.

Em seguida, estilizamos a classe CenteredContainer. Este contêiner flexbox também é centralizado, com largura e altura de 80%, e fundo branco. Adicionamos uma sombra ao redor do contêiner e configuramos a borda e o alinhamento de texto. Mudanças na cor de fundo demonstram como a centralização afeta a aparência visual do layout. A classe content é configurada com um padding de 20 pixels para ajustar o espaçamento interno.

Um exemplo prático de aplicação desse layout é uma página de login, onde o formulário precisa estar centralizado na tela para melhor usabilidade. Outro exemplo seria uma landing page de um produto, onde o conteúdo

principal deve ser destacado e centralizado para capturar a atenção do usuário de forma eficiente.

Layout fixo

Agora, vamos falar sobre o layout fixo. Nosso objetivo é entender os princípios básicos desse layout, como criá-lo utilizando HTML e CSS, os desafios que ele apresenta e as técnicas de CSS que podem ser aplicadas para melhorar sua implementação.

Para começar, o layout fixo é aquele em que a largura dos elementos é definida de maneira fixa e não muda conforme o tamanho da janela do navegador. Observe abaixo um exemplo prático de como criar um layout fixo utilizando HTML e CSS:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Layout Fixo</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Cabeçalho Fixo</h1>
    </header>
    <main>
        <div class="content">
```

```
            <h2>Conteúdo Principal</h2>
            <p>Este é um exemplo de layout fixo.</p>
        </div>
    </main>
    <footer>
        <p>Rodapé Fixo</p>
    </footer>
</body>
</html>
```

CSS (styles.css):

```
body {
    margin: 0;
    font-family: Arial, sans-serif;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
}
```

```
header, footer {
    width: 100%;
    background-color: #333;
    color: white;
    text-align: center;
    padding: 10px 0;
}
```

```css
main {
    flex: 1;
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100%;
}

.content {
    width: 600px;
    background-color: #f0f0f0;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

Esse layout fixo mantém a largura do conteúdo principal em 600 pixels, independentemente do tamanho da tela. Isso é útil para sites que requerem uma apresentação visual consistente, como blogs ou portfólios.

Isso pode ser útil para garantir que a aparência do site permaneça consistente em diferentes dispositivos. No entanto, um dos desafios desse tipo de layout é a falta de responsividade, o que pode prejudicar a usabilidade em telas menores.

Passando para a prática, vamos abrir o VS Code e a página renderizada pelo código HTML. O código que utilizamos está disponível no Git e pode ser baixado e aberto no VS Code para que você possa experimentar as alterações e ver os efeitos em tempo real. O nosso arquivo HTML segue o padrão que já vimos: a declaração do doctype, a tag HTML com a linguagem definida, e o head com metadados como charset, viewport e o título da página.

Dentro do body, que é a parte visível da nossa página, criamos um header com um título (h1), seguido pelo main, que contém o conteúdo principal da página. Dentro do main, utilizamos uma div com a classe content para estruturar o conteúdo, que inclui um título secundário (h2) e um parágrafo.

Por fim, adicionamos um footer. Sem o CSS, essa estrutura não se apresenta de forma organizada, mas o CSS é o que permite a centralização e a aplicação de estilos.

No arquivo CSS, definimos estilos para o body, removendo margens e definindo a altura, o display flex, o alinhamento centralizado e a direção em coluna. Isso garante que todos os elementos filhos sejam organizados verticalmente e centralizados na página. Estilizamos o header e o footer para ocupar 100% da largura, com cores de fundo e alinhamento centralizado do texto. Podemos alterar as cores para visualizar o efeito que isso causa.

Para o main, permitimos que o conteúdo principal cresça para ocupar o espaço disponível. Usamos display flex, alinhamento centralizado e justificativa centralizada. A classe content define a largura fixa de 600 pixels, com uma cor de fundo, padding, bordas arredondadas e sombra. Isso garante que o conteúdo principal fique fixo no centro da tela, independente do tamanho da janela.

Um exemplo prático de aplicação de um layout fixo seria um site de portfólio onde as imagens e descrições dos projetos precisam ter um tamanho específico para manter a consistência visual. Outro exemplo é um blog, onde o texto e as imagens são organizados em uma largura fixa para facilitar a leitura.

Layout elástico

Na última parte da nossa aula, abordaremos o layout elástico. Este tipo de layout é caracterizado pela sua capacidade de se adaptar ao conteúdo, ajustando-se conforme novos elementos são adicionados. Vamos explorar os fundamentos do layout elástico, sua implementação em HTML e CSS,

além de discutir suas vantagens e desvantagens, e como integrá-lo com outras técnicas de layout.

O layout elástico é ideal para situações onde o conteúdo da página pode variar em quantidade e tamanho, exigindo que o layout se ajuste dinamicamente para acomodá-lo de maneira adequada.

Veja um exemplo prático de como criar um layout elástico utilizando HTML e CSS:

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Layout Elástico</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Layout Elástico</h1>
    </header>
    <main>
        <div class="elastic-container">
```

```html
            <h2>Conteúdo Adaptável</h2>
            <p>Este layout ajusta-se ao conteúdo e ao tamanho da tela.</p>
        </div>
    </main>
    <footer>
        <p>Rodapé Elástico</p>
    </footer>
</body>
</html>
```

CSS (styles.css):

```css
body {
    margin: 0;
    font-family: Arial, sans-serif;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
}
```

```css
header, footer {
    width: 100%;
    background-color: #444;
    color: white;
    text-align: center;
    padding: 10px 0;
}

main {
    flex: 1;
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100%;
```

```css
}

.elastic-container {
    max-width: 80%;
    padding: 2em;
    background-color: #e0e0e0;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}
```

No exemplo acima, o ‘.elastic-container’ ajusta sua largura com base no conteúdo e nas unidades relativas (em/rem), o que o torna responsivo às

mudanças no tamanho da fonte e ao zoom do navegador. Isso é útil para páginas de leitura ou perfis de usuário que precisam se adaptar a diferentes tamanhos de conteúdo.

A implementação desse layout envolve o uso de unidades relativas no CSS, como em ou rem, que permitem que os elementos reajam às mudanças no tamanho da fonte.

Para exemplificar, vamos abrir o VS Code e analisar o código HTML e CSS que compõem o layout elástico. A estrutura HTML segue o padrão básico: temos a tag HTML, a seção head com metadados, o título da página e a ligação com o arquivo CSS. No body, criamos um header com um título (h1), seguido pelo main, que delimita a área principal do conteúdo. Dentro do main, uma div com a classe elastic-container organiza o conteúdo, incluindo um título secundário (h2) e parágrafos. Finalizamos com um footer.

O CSS é responsável por aplicar os estilos que tornam o layout elástico. Definimos seletores como body, header, main e footer, e aplicamos propriedades como font-family, margin, padding, display, justify-content e align-items. Utilizamos display: flex para criar um layout flexível, onde os elementos podem se ajustar dinamicamente.

A classe elastic-container é estilizada para ter uma largura máxima e mínima, permitindo que o conteúdo cresça ou diminua conforme necessário. Adicionamos um background-color, padding, sombra e bordas arredondadas para melhorar a estética. A centralização do texto e a margem entre os elementos garantem uma disposição harmoniosa e equilibrada.

Um exemplo prático de aplicação do layout elástico seria uma página de artigos, onde o conteúdo textual pode variar significativamente em comprimento. Outro exemplo é uma página de perfil de usuário, onde as informações podem mudar com frequência e precisam se ajustar sem comprometer o design da página.

O layout elástico apresenta vantagens, como a flexibilidade e a capacidade de adaptação a diferentes conteúdos e dispositivos. No entanto, também pode trazer desafios, como a necessidade de testes extensivos para garantir que o layout funcione corretamente em todas as situações.

Com isso, concluímos a nossa aula abordando os layouts centralizado, fixo e elástico. Cada tipo tem suas particularidades e aplicações ideais, e o conhecimento dessas técnicas é fundamental para a criação de interfaces web eficientes e atraentes.

Conteúdo Bônus

Para se aprofundar no tema desta aula, sugiro que assista ao vídeo intitulado "O QUE É UM SUBESPAÇO VETORIAL?: Definição, Testes e Exercícios Resolvidos | Álgebra Linear", que está disponível no canal Matemateca - Ester Velasquez no YouTube.

Referência Bibliográfica

BONATTI, D. **Desenvolvimento de Sites Dinâmicos com Dreamweaver CC.** Brasport: 2013.

BONATTI, D. **Desenvolvimento de Jogos em HTML5.** Brasport: 2014.

FLATSCHART, F. HTML 5 - **Embarque Imediato.** Brasport: 2011.

JOÃO, B. do N. (Org.). **Informática aplicada.** 2.ed. Pearson: 2019.

MARINHO, A. L.; CRUZ, J. L. da. **Desenvolvimento de aplicações para Internet.** 2.ed. Pearson: 2020

NEVES, M. C. B. de A. **Sites de Alta Performance.** Contentus: 2020

SOUSA, R. F. M. CANVAS HTML 5 - **Composição gráfica e interatividade na web.** Brasport: 2018.

TANENBAUM, A. S.; FEAMSTER, N.; WETHERALL, D. J. **Redes de computadores.** 6.ed. Pearson: 2021.

Ir para exercício