Design Responsivo

Link Notion: https://cherry-client-b8f.notion.site/Design-Responsivo-177911d84e0d80da9536dd6af352063b?pvs=73

O design responsivo é uma abordagem de desenvolvimento web que visa criar sites que se adaptam dinamicamente ao tamanho da tela e ao dispositivo do usuário, garantindo uma experiência consistente e otimizada em qualquer dispositivo, seja ele um computador desktop, tablet ou smartphone.

Comece pelo tamanho da tela

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Essa meta tag informa ao navegador como deve se comportar em dispositivos móveis, controlando o **tamanho da área visível (viewport) e o nível de zoom inicial**.

`<meta name="viewport" ...>`

- É uma **meta tag HTML** usada para fornecer **instruções ao navegador** sobre como dimensionar e escalar a página.
- A **viewport** é a **área visível do site na tela do dispositivo**.

`width=device-width`

- Isso define que a **largura da viewport** deve ser igual à **largura real do dispositivo**.
- Ou seja, em um celular com tela de 360px, a viewport terá 360px.

Sem isso, a maioria dos navegadores móveis assume uma largura fictícia de **980px**, e "diminui" a página inteira para caber — o que causa:

- texto pequeno,
- layout quebrado,
- zoom forçado.

`initial-scale=1.0`

- Define o **nível de zoom inicial da página** como **100% (1.0)**.
- Isso significa que a página não será aumentada nem reduzida quando for carregada.

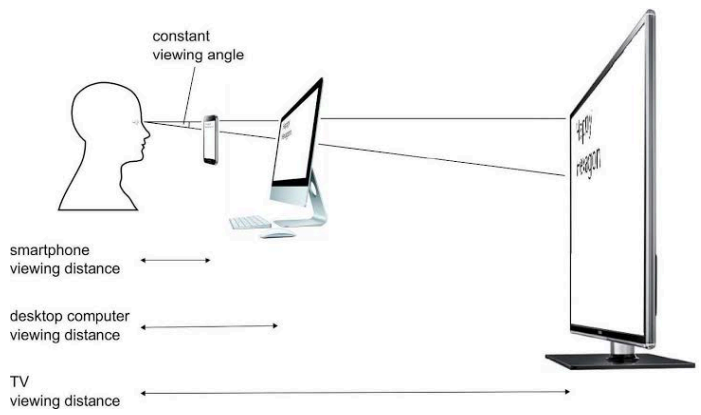
Você também pode usar:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
```

Atributo	Significado
`maximum-scale=1.0`	Impede o usuário de aumentar o zoom
`user-scalable=no`	Desativa o gesto de zoom com os dedos

Texto responsivo

O tamanho do texto deve ser maior conforme a distância que iremos consumi-lo. Quanto maior a distância, maior o texto.



Atualmente a melhor forma de lidar com isso, é utilizando o **clamp()**.

Clamp()

O clamp() é uma função nativa do CSS usada para definir valores responsivos e limitados por mínimo e máximo — ideal para fontes, tamanhos de elementos, margens, etc.

Ela permite que você combine valores fixos e flexíveis de forma elegante, sem precisar de media queries.

📦Sintaxe

```
clamp(min, ideal, max)
```

Exemplo com Media Query

Aqui, você define “faixas” específicas e ajusta manualmente.

```
h1 {
  font-size: 1.5rem;

  @media (min-width: 600px) {
    font-size: 2.5rem;
  }

  @media (min-width: 1000px) {
    font-size: 3rem;
  }
}
```

Exemplo com clamp()

```
h1 {
  font-size: clamp(1.5rem, 4vw, 3rem);
}
```

Aqui, você define tudo em uma única linha:

- Nunca será menor que 1.5rem.
- Cresce com 4vw (4% da largura da tela).
- Nunca passa de 3rem.

💡Dica

Existem sites que calculam o valor ideal do clamp como o https://royalfig.github.io/fluid-typography-calculator/

Ajustando imagens

É sempre recomendado, iniciar colocando a **largura máxima** da imagem em 100%, para evitar que ela ultrapasse o tamanho do container e cause rolagem horizontal e **altura como auto**, para que a altura se ajuste automaticamente baseado na largura, redimensionando de forma proporcional.

```
<img src="img_girl.jpg"
style="max-width:100%; height:auto;">
```

É importante carregar imagens o mais leve possível, pensando na velocidade de carregamento e nos usuários do site mobile que utilizam a rede móvel para acessar.

Uma das opções, é a tag **<picture>** do HTML, onde é possível determinar imagens diferentes para diferentes breakpoints.

```
<picture>
  <source srcset="img_smallflower.jpg" media="(max-width: 600px)">
  <source srcset="img_flowers.jpg" media="(max-width: 1500px)">
  <source srcset="flowers.jpg">
  <img src="img_smallflower.jpg" alt="Flowers">
</picture>
```

Evite tamanhos fixos

Utilize medidas relativas como %, em, rem, vh, dvh e propriedade como min-width e min-height, para que o layout possa se adaptar à diferentes contextos.

1. **px(pixels)**

O que é: Unidade fixa, absoluta.

- 1px = 1 ponto na tela.

Quando usar:

- Bordas, sombras e ícones que precisam ter tamanho exato.
- Elementos que não devem escalar com o texto ou viewport.

2. **% (porcentagem)**

O que é: Unidade relativa ao elemento pai.

Quando usar:

- Largura e altura de containers (width, height).
- Posicionamento relativo (top, left, margin, padding).

3. **em**

O que é: Unidade relativa ao tamanho da fonte do elemento pai.

- 1em = 100% do font-size do pai.

Quando usar:

- Padding, margin e fontes que devem escalar proporcionalmente ao texto do container.
- Boas práticas de acessibilidade, quando o usuário altera o tamanho da fonte.

4. **rem**

O que é: Unidade relativa ao tamanho da fonte raiz (html), geralmente 16px.

- 1rem = 16px (padrão), mas muda se html { font-size } for alterado.

Quando usar:

- Fontes e layout consistentes em todo o site, independente do nesting do elemento.
- Componentes reutilizáveis e escaláveis.

5. **vw/vh(viewport width / height)**

O que é: Percentual da largura ou altura da viewport.

- 1vw = 1% da largura da tela
- 1vh = 1% da altura da tela

Quando usar:

- Layouts full screen (hero sections, banners).
- Fontes responsivas que crescem ou diminuem com a tela.

6. **ch**

O que é: Unidade baseada na largura do caractere “0” da fonte atual.

Quando usar:

- Para comprimento de inputs e textareas.
- Quando se quer um layout proporcional ao texto.

7. **dvw/dvh(Dynamic Viewport Width / Height)**

O que é:

- Similar a vw/vh, mas considera mudanças dinâmicas da viewport, como barras de endereço em mobile que aparecem ou desaparecem.
- 1dvw = 1% da largura da viewport dinâmica
- 1dvh = 1% da altura da viewport dinâmica

Quando usar:

- Layouts full screen em mobile, que precisam se adaptar quando a barra de navegação aparece/desaparece.
- Hero sections e modais que devem ocupar toda a altura visível.

8. **svw/svh(Small Viewport Width / Height)**

O que é:

- Unidade que considera a menor viewport possível, útil para layouts que precisam garantir visibilidade mínima do conteúdo.
- 1svw = 1% da menor largura da viewport
- 1svh = 1% da menor altura da viewport

Quando usar:

- Layouts críticos, como banners e botões, que devem ser visíveis mesmo quando a tela é reduzida temporariamente (ex.: modo split screen ou rotacionamento de tela).
- Evita que o conteúdo fique cortado em dispositivos com viewports variáveis.

Neste vídeo é possível entender um pouco melhor em quais casos cada medida é mais utilizada.

https://youtu.be/N5wpD9Ov_To?si=GHCDFGWapwOSHD3b

Para problemas complexos, utilize Media Queries

Media queries são um recurso do CSS que permitem que um site adapte seu layout e estilos dependendo das características do dispositivo do usuário, como a largura da tela, a orientação (retrato ou paisagem) e a resolução.

Estrutura básica

```
@media <tipo> and (<condição>) {
  /* estilos aplicados se a condição for verdadeira */
}
```

Tipo

Podemos uitlizar os valores:

- **all** - Aplica os estilos para todos os tipos de tela
- **screen** - Telas (monitores, celulares, tablets)
- **print** - Impressoras ou visualização de impressão (Ctrl+P)
- **speech** - Leitores de tela que transformam o conteúdo em voz.

Largura de tela

A largura de tela é definida por **breakpoints**, que são pontos de interrupção que definem quando o layout de um site é alterado para se adaptar a diferentes tamanhos de tela ou dispositivos. Abaixo está o exemplo de aplicação mobile-first utilizada no site da W3Schools.

```
@media only screen and (max-width: 600px) {...}

/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {...}

/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {...}

/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {...}

/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {...}
```

Orientação

Podemos utilizar dois valores para orientação: portrait(retrato ou celular na vertical) ou landscape(paisagem ou celular na horizontal). Podemos utilizar operadores como **and, or ou not** para combinar condições**.**

```
@media (min-width: 600px) and (orientation: landscape) {
  .sidebar {
    display: block;
  }
}
```

Container Query

Container Queries são uma funcionalidade moderna do CSS que permite aplicar estilos com base **no tamanho de um elemento pai (container)**, e não no tamanho da **janela (viewport)** — como acontece com media queries.

O primeiro passo é definir o container pai com **container-type**.

```
.sidebar {
  container-type: inline-size; /* ou size para largura + altura */
  container-name: sidebar;
}
```

Depois podemos mencionar o container através do nome e aplicar uma regra baseada no tamanho do container.

```
@container sidebar (min-width: 300px) {
  .menu-item {
    font-size: 1.2rem;
  }
}
```

Grid responsivo

Para criar grids responsivos, nós podemos utilizar duas propriedades em conjunto: **minmax + auto-fit/autofill**.

Minmax

O **minmax** define um tamanho mínimo e máximo para uma coluna ou linha do grid, permitindo que a coluna ou linha cresça ou encolha dentro de limites definidos.

```
grid-template-columns: repeat(3, minmax(150px, 1fr));
```

Neste caso, cada coluna terá no mínimo **150px** e no máximo, ocupará o **restante do espaço disponível (1fr).**

Usos comuns

- Layouts de cards ou imagens que precisam de largura mínima para não quebrar o conteúdo, mas que devem crescer para ocupar o container.
- Flexibilidade de colunas sem precisar criar media queries para cada breakpoint.

Auto-fit

O grid cria colunas apenas para os elementos presentes, e as colunas vazias são “esmagadas” a 0, fazendo com que os elementos existentes se expandam para ocupar todo o espaço disponível.

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

Com 3 elementos em um container grande, cada coluna cresce para preencher o espaço restante, sem criar colunas extras vazias.

Usos comuns

- Grid de cards ou fotos onde queremos que o conteúdo se expanda e ocupe todo o container disponível.
- Útil quando o número de elementos é dinâmico, evitando espaços vazios.

Auto-fill

Com o auto-fill, o grid cria quantas colunas couberem no container, preenchendo também colunas vazias se sobrar espaço.

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
```

Neste caso, o grid cria o número máximo de colunas de 200px no mínimo, mas mesmo que haja menos elementos, ele **continua ocupando o espaço** criando colunas “vazias”, mantendo o layout uniforme, com colunas sempre iguais.

Usos comuns

- Grid de cards onde o número de colunas deve se manter constante.
- Mantém alinhamento uniforme mesmo com poucos elementos.

Referências

https://youtu.be/vQDgoQKfdzM?si=idibcZPRbJPBDUU2

https://youtu.be/rrLAg7xNERA?si=W9juJOvP7ZOvzNTy

https://www.youtube.com/watch?v=srvUrASNj0s&t=712s

https://www.w3schools.com/html/html_responsive.asp