Documentação SCSS

Aprenda as funcionalidades CSS básicas do framework e como essa biblioteca pode otimizar e acelerar a produção do front-end da sua aplicação.

Introdução

Conceitos básicos de como são aplicadas as variáveis, constantes, classes auxiliares, funções de classes, mixins entre outras funcionalidades da biblioteca.

Boas Práticas com as classes utilitárias do WPF

Com seu projeto HTML definido e o WPF3 importado corretamente, começar a recortar a sua página web é de longe a etapa mais fácil de todo o processo.

Apesar de não existir uma lógica definida ou uma forma obrigatória de organizar as suas classes dentro do atributo *class=””* na sua tag HTML, existe uma ordem que pessoalmente recomendamos para que você siga e se acostume, para garantir que seu código permaneça legível e organizado, mantendo um único padrão constante entre as páginas.

É importante lembrar que o WPF3 foi desenvolvido tendo em mente um escopo de projeto que é baseado em **componentes** e, por mais que ele seja ótimo para o recorte e desenvolvimento web tradicional, é através de projetos que utilizam a estruturação MVC que o WPF realmente brilha.

Sugerimos sempre você separe suas classes na seguinte ordem:

<div class=”*custom* func() *pos display conf. display tam orient extra*”>

Vamos entrar em mais detalhes sobre como essa organização funciona:

- **Custom:** Aqui vão todas as propriedades e classes criadas por você que não fazem parte do escopo do WPF. A ideia é que seja fácil visualizar o que é feito especificamente por você para incrementar os estilos que devem ser aplicados somente à uma ocasião específica;
- **Funções:** Quaisquer **funções de classe** são atribuídas logo depois. Essas classes especiais costumam configurar cores de todos os tipos (plano de fundo, borda, texto, entre outras) ou até mesmo definem estruturações visuais básicas (como a família ou peso de uma fonte que uma porção de texto terá);
- **Posicionamentos:** Nessa parte é onde inserimos as primeiras classes específicas do WPF. Aqui vamos tratar de posicionamentos ou tratamentos de posição, como todas as classes que envolvem as propriedades *position, box-sizing* entre outros similares;
- **Display:** Agora vamos especificar o tipo de exibição que seu elemento terá: *grid, table, block, inline-block, contents, flex* entre outros;
- **Conf. do Display:** E depois damos os detalhes e configurações disponíveis dessa exibição específica, como é o caso de *align-items, justify-block, vertical-align* entre outros;

- **Tamanho e Espaçamento:** Definimos então o tamanho do nosso elemento com *width* e *height*, além dos espaçamentos disponíveis, como *margin* e *padding*;
- **Orientações:** Por último, caso sua tag HTML necessite, aqui entra todos os tipos de orientações, como orientação de texto em *text-align* ou denominar um espaço específico do elemento para não renderizar mais linhas de texto do que o instruído, como *clamp* etc;
- **Extra:** Se sua tag HTML necessita de outras classes que fazem parte da estrutura do WPF, mas não entram em nenhuma dessas categorias, como *transition, transform, filter,* entre outras, você pode livremente alocá-las em qualquer ordem após ter organizado tudo seguindo este roteiro. As classes extras também abordam todas as classes que você precisa inserir para substituir um valor padrão que você atribuiu anteriormente. Nesse caso, uma classe de qualquer uma das outras categorias usando a instrução "!" (como por exemplo, uma que possuí valor padrão w-50 mas que precisa receber um valor novo de w-100 ou w-25 receberia o valor como !w-100 ou !w-25)

Lembrando que não é obrigatório utilizar essa organização específica. Ela se trata apenas de uma sugestão. Sinta-se encorajado a encontrar a melhor forma que funciona para você e sua equipe.

Atribuições de Responsividade, Importância e Pseudo Elementos

Todas as classes do WPF são compatíveis com o que chamamos de **atribuições**. As **atribuições** são nada mais do que comandos extras que podem ser inclusos em uma classe utilitária do WPF que impõe sobre ela comportamentos extras ou específicos. Esses comportamentos podem ser de inúmeros tipos: *responsivos, importantes ou pseudos.*

Nesta lição, veremos um pouco mais sobre cada um deles e como podemos atribuir cada uma dessas **atribuições** nas nossas classes auxiliares:

Responsividade

As **atribuições de responsividade** podem ser todas encontradas no arquivo de instruções do WPF, no arquivo `*wpf.config.yaml*` dentro do seu projeto. Na folha de estilos do `wpf.config.yaml` você pode encontrar uma seção chamada "**breakpoints**":

image.png

Cada um dos itens listados sobre essa lista refere-se à uma regra de estilo que define uma nova resolução em largura (utilizando pixels) para definir um ponto de quebra na renderização da sua página. Pegue '*mob: 1024*' por exemplo: Se você atribuir o prefixo **"mob:"** sob sua classe utilitária, antes do nome da classe, você estará definindo um estilo que será aplicado automaticamente em dispositivos que tenham uma **largura menor do que 1024px**. O mesmo vale para todas as outras variações definidas sob essa lista.

Por exemplo, ao utilizar a combinação de classes `**flex**` e `**mob:grid**` em um elemento `**<div>**`, você pode alcançar resultados interessantes:

```
<div class="dp-flex mob:dp-grid">
    <!-- Conteúdo aqui -->
</div>
```

Neste exemplo, o elemento `<div>` terá o estilo `display: flex` em contextos padrão do projeto, mas quando a largura da tela for menor que 1024px, o estilo será alterado para `display: grid`, permitindo uma transição suave e eficiente entre layouts responsivos.

Aqui o resultado final, em um CSS padrão, ficaria assim:

```
.dp-flex{
  display: flex;
}

@media (max-width: 1024px){
  .mob\:dp-grid{
    display: grid;
  }
}
```

Se for necessário atribuir novos tipos de breakpoints para o seu projeto, você pode editar essa lista de breakpoints presentes no *wpf.config.yaml* para criar seus próprios pontos de quebra customizados para designs responsivos.

Importância

As **atribuições de importância** servem para definir, em um código CSS onde um elemento possuí duas vezes o mesmo atributo sob situações conflitantes, um ponto de partida para que o CSS descubra qual regra deve sempre se sobressair como o estilo dominante.

Leve por exemplo um elemento que precisa mudar o seu tamanho sem necessariamente estar sob um breakpoint. Ele tem um tamanho original de 50% (informado pela classe utilitária `w-50%`). Esse mesmo elemento receberá do JavaScript uma nova classe que mudará o seu tamanho para 25% (informado pela classe utilitária `w-25%`) quando o usuário clicar em algum botão dentro do site. Como são elementos conflitantes (a largura, no caso), dificilmente o navegador saberá entender que o valor de 25% é o correto a ser aplicado agora. Informando o código que o novo estilo **é mais importante** (atribuindo a classe utilitária com o modificador de importância `!w-25%`) agora o navegador saberá o momento em que o elemento precisa ter 25% de largura e o momento em que ele precisa ter 50%.

```
<div x-data="{ isNarrow: false }">
  <div class="w-50%" x-bind:class="{ '!w-25%': isNarrow }">
     <!-- Conteúdo aqui -->
  </div>
</div>
```

O resultado final disso, em um arquivo CSS comum, seria bem simples:

```
.w-50\%{
  width: 50%;
}

.\!w-25\%{
  width: 25%!important;
}
```

É possível adicionar esse tipo de comportamento à elementos que estão com **atribuições de responsividade**. Veja um exemplo:

```
<div x-data="{ isNarrow: false }">
  <div class="mob:w-50%" x-bind:class="{ 'mob:!w-25%': isNarrow }">
     <!-- Conteúdo aqui -->
  </div>
</div>
```

Pseudo Elementos

Por último, existem as atribuições que chamamos normalmente no CSS como **pseudo elementos**. Isso implica a todos os elementos virtuais que podem ser gerados pelo *DOM* via CSS, como **:active, ::placeholder, ::after, ::before** etc. No WPF, alguns desses pseudo elementos possuem um suporte nativo pela ferramenta, e podem receber estilos e classes diretamente informando o pseudo-elemento que você deseja ativar/inserir para aquele elemento HTML.

Para definir um pseudo elemento à um elemento HTML é muito simples: basta você escrevê-lo antes da classe e de qualquer outra atribuição adicional que esteja inserindo à uma classe utilitária WPF. Daremos um exemplo:

```
<div class="bg(c-bg-1) hover:bg(c-bg-2)">
   <!-- Conteúdo aqui -->
</div>
```

No exemplo acima, o elemento possuí a cor de background `var(--c-bg-1)` que é automaticamente alterada para `var(--c-bg-2)` no momento em que o usuário passar o cursor do mouse por cima do elemento, ou seja, através da ativação do pseudo elemento **:hover**. Esses tipos de atribuições não geram um resultado diretamente no CSS, mas são lidos de maneira dinâmica diretamente no navegador.

É importante lembrar que as **atribuições de pseudo elementos funcionam em conjunto com os outros tipos de atribuições, como as de responsividade ou importância**. Veja um exemplo:

```
<div class="bg(c-bg-1) mob:bg(c-bg-2) hover:mob:bg(c-bg-3)">
    <!-- Conteúdo aqui -->
</div>
```

No exemplo acima, a versão natural do componente não possuí um comportamento com o pseudo elemento **:hover**. Esse comportamento foi instruído para acontecer somente em um contexto responsivo, com duas outras cores completamente diferentes.

Declarando e Usando Variáveis CSS

As variáveis customizadas de cada projeto são inseridas dentro de uma pasta chamada **"vars"**. Todos os arquivos nessa pasta são automaticamente incluídos como variáveis no framework. No entanto, existe um arquivo específico dentro dessa pasta chamado `_default.scss`, onde ***variáveis especiais podem ser declaradas***.

É importante frisar que **variáveis especiais precisam ser declaradas entre aspas simples**. Isso é exigido não só pelo manual de boas práticas do próprio SASS como também porque essas variáveis passam por vários tratamentos específicos e não funcionarão de outra maneira.

Sass Guidelines

Grupos de Variáveis

As variáveis são organizadas em grupos específicos pré-determinados para facilitar o gerenciamento e a aplicação consistente em todo o projeto. Alguns dos grupos comuns incluem:

Cores

As cores são frequentemente organizadas em grupos separados com diferentes propósitos, como cores de fundo, cores de texto, cores de borda, cores de gradiente, etc. Aqui está um exemplo de como os grupos de cores podem ser declarados:

```
//* Cores de Fundo
$c-backgrounds: (
    "c-bg-1": '#141616',
    "c-bg-2": '#FFFFFF',
);

//* Cores de Texto
$c-strings: (
```

```
    "c-s-1": '#FFFFFF',
    "c-s-2": '#999999',
);

//* Cores de Bordas
$c-borders: (
    "c-border-1": '#444444',
    "c-border-2": '#666666',
);
```

Temas

Os temas podem incluir uma variedade de cores específicas de design, como cores principais, secundárias, acentuadas, etc. Aqui está um exemplo de como os temas podem ser declarados:

```
//* Cores do Tema
$c-themes: (
    "c-t1-050": #effcfc,
    "c-t1-100": #d8f5f4,
    "c-t1-400": #42c1c7, //? Main
);
```

Uso das Variáveis

Depois de declaradas as variáveis, elas podem ser facilmente utilizadas em todo o projeto por meio de classes específicas. Por exemplo, para usar uma cor de fundo específica em um elemento HTML, podemos adicionar a classe correspondente:

```
<div class="bg(c-bg-1)"></div>
```

Se quisermos aplicar estilos diferentes para dispositivos móveis, podemos adicionar classes específicas para isso:

```
<div class="bg(c-bg-1) mob:bg(c-bg-2)"></div>
```

Isso permite que tenhamos controle granular sobre a aparência de elementos em diferentes dispositivos.

Funções de Classe

Além das classes específicas para aplicação de variáveis, também existem funções de classe para casos comportamentais diferentes. Por exemplo:

- Alterar a fonte de alguma maneira: **`fnt(var)`**
- Alterar a cor de fundo: **`bg(var)`**
- Alterar a cor de texto: **`c(var)`**
- Alterar a cor de borda: **`bc(var)`**
- Adicionar efeitos visuais: `fx(param,var)` / `fx(var)`

Essas funções de classe permitem uma aplicação mais flexível e eficiente das variáveis CSS em todo o projeto.

Usando as variáveis dentro de outros arquivos SCSS

Todas as variáveis declaradas no arquivo `_default.scss` geram uma versão de variável CSS comum. Essa versão permite o uso das variáveis dentro de outros arquivos CSS do projeto.

Por exemplo, a variável **`c-bg-1`**, que pode ser diretamente usada no HTML como **`bg(c-bg-1)`**, também pode ser utilizada no CSS da seguinte maneira:

```
background-color: var(--c-bg-1);
```

Isso oferece flexibilidade adicional para o uso das variáveis em diferentes contextos dentro do projeto.

Classes Utilitárias

Aprenda a usar todas as classes utilitárias auxiliares do WPF, desde o seu conceito até a aplicação.

Position

O atributo **position** é retratado de uma maneira bem natural e intuitiva pelo WPF. Para atribuir classes utilitárias de posição no seu código, basta escrever o nome do valor do atributo diretamente como uma classe css:

```
<div class="relative">
  <!-- O conteúdo dessa classe recebeu o atributo css 'position: relative' -->
</div>

<div class="absolute">
  <!-- O conteúdo dessa classe recebeu o atributo css 'position: absolute' -->
</div>

<div class="sticky">
  <!-- O conteúdo dessa classe recebeu o atributo css 'position: sticky' -->
</div>

<!-- Isso funciona para qualquer outro tipo de atribuição de posicionamento. -->
```

Abaixo segue uma lista com todas as posições suportadas pelo framework:

Classe Utilitária	Comportamento
`static`	*Posicionamento padrão, segue o fluxo normal do documento.*
`sticky`	*Alterna entre* `relative` *e* `fixed`*, fixando o elemento ao atingir um ponto de rolagem definido.*
`relative`	*Posicionado relativo à sua posição original. Permite ajus* `s` `t` `t` `nt`, `tom` e `left`.
`absolute`	*Posicionado em relação ao ancestral mais próximo com po* `o não` `static`*, sai do fluxo normal.*
`fixed`	*Posicionado em relação à janela do navegador, permanece fixo durante a rolagem.*

Offset

As classes utilitárias de offsets no **WPF** permitem ajustar a posição de elementos com as propriedades CSS top, right, bottom e left. Elas são projetadas para serem intuitivas, seguindo uma sintaxe clara e flexível, compatível com unidades como `%`, `px`, `dvh` e `dvw`.

Como Funcionam as Classes de Offsets

As classes de offsets seguem o padrão: (propriedade)(valor)(unidade), onde:

- **Propriedade**: `top`, `right`, `bottom` ou `left`.
- **Valor**: Um número inteiro (positivo ou negativo) que define o deslocamento.
- **Unidade**: Opcional, pode ser `%`, `px`, `dvh` (altura dinâmica da viewport) e `dvw` (largura dinâmica da viewport). Se omitida, o valor é tratado como unitless (ex.: `top(10)` equivale a `top: 10;`).

Isso significa que uma classe válida segue este formato:

```
<div class="fixed top(50%) left(20px)">
  <!-- Esse elemento html definiu os seguintes atributos css - top: 50%; left: 20px; -->
</div>

<div class="absolute right(-10dvw) bottom(100)">
  <!-- Esse elemento html definiu os seguintes atributos css - right: -10dvw; bottom: 100; -->
</div>
```

Usando Funções de Classe para Centralização

As classes utilitárias de centralização do **WPF** simplificam o alinhamento de elementos horizontalmente, verticalmente ou em ambos os eixos. Elas combinam as propriedades left, top e transform para posicionar elementos com precisão, especialmente em elementos com position: `absolute`, `fixed` ou `sticky`.

Como Funcionam as Classes de Centralização

As classes de centralização seguem uma sintaxe simples e intuitiva: **center(tipo)**, onde o tipo define o eixo de centralização:

As classes de centralização **são ideais para elementos com position:** `absolute`, `fixed` ou `sticky`, **dentro ou fora de um contêiner pai com position semelhante**. Certifique-se de que o elemento pai tenha dimensões definidas de alguma maneira (ex.: width e height) para que a centralização funcione corretamente.

Classe Utilitária	Comportamento
`center(h)`	*Centraliza horizontalmente o elemento.*
`center(v)`	*Centraliza verticalmente o elemento.*
`center(c)`	*Centraliza em **ambos** os eixos (horizontal e vertical).*

Com essa sintaxe, confira alguns exemplos dessas funções de classe em execução:

```
<div class="fixed top(20px) center(h)">
  <!-- No seguinte elemento html, a seguinte configuração css foi atribuída - top: 20px; left: 50%; transform: translateX(-50%); -->
</div>

<div class="absolute top(5%) center(v)">
  <!-- No seguinte elemento html, a seguinte configuração css foi atribuída - left: 5%; top: 50%; transform: translateY(-50%); -->
</div>

<div class="relative center(c)">
  <!-- No seguinte elemento html, a seguinte configuração css foi atribuída - left: 50%; top: 50%; transform: translate(-50%, -50%); -->
</div>
```

Notas

> **Posicionamento**: As classes de centralização requerem que o elemento tenha position: `absolute`, `fixed` ou `sticky`.

> **Contêiner Pai**: Certifique-se de que o elemento pai tenha algum atributo position não estático e dimensões definidas de alguma maneira.

>

> **Transformações Adicionais**: Certifique-se que, ao usar outros tipos de **configurações de transformação**, que elas não sobrescrevam os seus estilos!

Z-Index

As classes utilitárias de z-index do **WPF** permitem controlar a ordem de empilhamento de elementos no eixo Z, definindo qual elemento aparece acima ou abaixo de outros. Elas são ideais para layouts com sobreposições, como modais, menus ou elementos posicionados com absolute ou fixed.

Como Funcionam as Classes de Z-Index

As classes de z-index seguem o padrão: **z-(valor)**, onde:

- **Valor**: Um número inteiro (positivo ou negativo) que define a ordem de empilhamento.
- **Sintaxe**: A classe é formada por *z-* seguido de um número, como `z-10`, `z--5` ou `z-0`.

Regras CSS Geradas

Cada classe é traduzida diretamente para a propriedade CSS z-index. Por exemplo:

- `z-10` → `z-index: 10;`
- `z--5` → `z-index: -5;`
- `z-0` → `z-index: 0;`

Como Usar

As classes de **z-index** são aplicadas a elementos com position: `absolute`, `relative`, `fixed` ou `sticky`, já que o z-index só afeta elementos posicionados. Use-as para controlar a sobreposição de elementos em layouts complexos.

Exemplo 1: Sobreposição Simples

Posicione um elemento acima de outro:

```
<div class="relative w-100% h-200px">
  <div class="absolute top(0) left(0) bg(c-blue) z-10"><!-- Elemento superior --></div>
  <div class="absolute top(10px) left(10px) bg(c-red) z-5"><!-- Elemento inferior --></div>
</div>
```

Exemplo 2: Z-Index Negativo

Coloque um elemento abaixo do conteúdo padrão:

```
<div class="relative w-100% h-200px">
  <div class="absolute top(0) left(0) bg(c-gray) z--1"><!-- Fundo --></div>
  <div class="absolute top(20px) left(20px) bg(c-white)"><!-- Conteúdo principal --></div>
</div>
```

Notas

> **Valores altos**: Use valores como `z-100` ou superiores para elementos que devem ficar **acima de tudo**, como *modais*.

> **Valores negativos**: Use `z--1` ou **inferiores** para **elementos de fundo ou camadas abaixo do conteúdo principal**.

> O **z-index** só funciona em elementos com position diferente de `static`.

> Valores maiores de **z-index** colocam o elemento mais acima na ordem de empilhamento.

> Evite valores extremamente altos (ex.: `z-999999`) para **manter a manutenção do código simples**.

Overflow

As classes utilitárias de `overflow` do **WPF** controlam como o conteúdo excedente de um elemento é tratado, permitindo gerenciar a visibilidade e o comportamento de rolagem em elementos com dimensões limitadas. Elas são ideais para layouts com *contêineres*, *modais* ou áreas de conteúdo **restritas**.

Como Funcionam as Classes de Overflow

As classes de `overflow` seguem uma sintaxe intuitiva que permite definir o comportamento geral ou por eixo **(x ou y)**, com ou sem tipo específico (`hidden`, `scroll`, `visible`). O padrão `overflow` sem modificadores aplica `overflow: auto;` como valor padrão.

Atributos Válidos	Comportamento
`auto`	*Deixa o navegador encarregado do comportamento. Seu funcionamento pode variar.*
`hidden`	*O conteúdo é cortado e nenhuma barra de rolagem é exibida.*
`scroll`	*O conteúdo é acessível através de barras de rolagem que são exibidas mesmo que o conteúdo não precise.*
`visible`	***Valor padrão**. O conteúdo não é cortado e pode ser renderizado para fora da caixa de conteúdo.*

As classes são traduzidas diretamente para propriedades CSS. Veja alguns exemplos:

- `overflow`→ `overflow: auto;`
- `overflow-x` → `overflow-x: auto;`
- `overflow-y-hidden` → `overflow-y: hidden;`
- `overflow-scroll` → `overflow: scroll;`

Como Usar

As classes de `overflow` são aplicadas a elementos com dimensões definidas (ex.: `width`, `height`, `max-width`, `max-height`) para controlar o comportamento do conteúdo que excede essas dimensões. Elas são úteis para criar áreas roláveis, ocultar conteúdo excedente ou manter visibilidade.

Exemplo 1: Overflow Padrão

Aplica rolagem automática para conteúdo excedente em ambas as direções:

```
<div class="w-300px h-200px overflow">
  <p><!-- Conteúdo longo que excederá o contêiner... --></p>
</div>
```

Exemplo 2: Overflow por Eixo

Habilita rolagem horizontal, mas oculta o conteúdo vertical excedente:

```
<div class="w-300px h-200px overflow-x overflow-y-hidden">
  <p><!-- Conteúdo longo horizontalmente e verticalmente... --></p>
</div>
```

Exemplo 3: Overflow com Tipo Específico

Cria uma área com rolagem vertical forçada e conteúdo horizontal visível:

```
<div class="w-300px h-200px overflow-y-scroll overflow-x-visible">
  <p><!-- Conteúdo com rolagem vertical e visibilidade horizontal... --></p>
</div>
```

Exemplo 4: Combinação com Posicionamento

Usa overflow com classes de posicionamento e centralização:

```
<div class="relative w-100% h-400px overflow-hidden">
  <div class="absolute center(c) w-200px h-200px overflow-y-scroll">
    <p><!-- Conteúdo rolável centralizado... --></p>
  </div>
</div>
```

Notas

> **Dimensões**: Sempre defina `width`, `height`, `max-width` ou `max-height` no elemento para que o `overflow` funcione corretamente.

> **Rolagem Automática**: Use `overflow` ou `overflow-(x|y)` para rolagem apenas quando necessário.

>

Combinações: Combine com classes de posicionamento (`absolute`, `relative`, etc.) e offsets (`top`, `left`, etc.) para layouts complexos.

Forçar Rolagem: Aplicar o tipo `scroll` em um elemento evita o problema de barras de rolagem aparecendo e desaparecendo **quando o conteúdo é dinâmico**. Tenha em mente que **Impressoras podem imprimir o conteúdo vazado**.

As classes `overflow-x` e `overflow-y` são úteis para controle granular em layouts responsivos.

O valor `visible` pode causar sobreposição de conteúdo fora do contêiner, então use com cuidado.

Para áreas roláveis, certifique-se de que o contêiner tenha conteúdo suficiente para ativar a rolagem.

Text-Align

Os alinhadores de texto do **WPF** são classes utilitárias que controlam o alinhamento horizontal de texto em elementos HTML. Eles utilizam a propriedade CSS `text-align` para oferecer flexibilidade e simplicidade no ajuste de layouts textuais.

Como Funcionam os Alinhadores de Texto

Os alinhadores seguem as seguintes definições principais:

Classe Utilitária	Comportamento
`t-left`	*Alinha o texto à esquerda.*
`t-center`	*Centraliza o texto.*
`t-right`	*Alinha o texto à direita.*
`t-start`	*Alinha o texto no início da **direção configurada do texto** .*
`t-end`	*Alinha o texto no final da **direção configurada do texto**.*
`t-justify`	*Justifica o texto.*
`t-justify-all`	*Justifica todas as linhas do texto, **incluindo a última**.*

Exemplos de Uso

Alinhamento Básico

```
<p class="t-left"><!-- Texto à esquerda. --></p>
<p class="t-center"><!-- Texto centralizado. --></p>
<p class="t-right"><!-- Texto à direita. --></p>
```

Alinhamento Dinâmico

```
<p class="t-start"><!-- Texto no início (esquerda em LTR). --></p>
<p class="t-end"><!-- Texto no final (direita em LTR). --></p>
```

Justificação

```
<p class="t-justify"><!-- Texto justificado, exceto a última linha. --></p>
<p class="t-justify-all"><!-- Texto justificado, incluindo a última linha. --></p>
```

Notas

> **Suporte a Idiomas**: Use `t-start` e `t-end` para layouts que precisam se adaptar a direções de texto como *RTL (right-to-left)*.

> **Justify vs. Justify-All**: `t-justify` é ideal para parágrafos normais, enquanto `t-justify-all` é útil para textos curtos que exigem alinhamento completo.

> O valor `justify-all` pode **não ser suportado** em navegadores mais antigos; teste a compatibilidade se necessário.

Word-Break

As classes utilitárias de `word-break` no WPF são projetadas para controlar o comportamento de quebra de palavras em elementos HTML, utilizando a propriedade CSS `word-break`. Elas são ideais para gerenciar textos longos, layouts responsivos ou conteúdos em idiomas que não utilizam espaços entre palavras.

Como Funcionam as Classes de Word-Break

As classes seguem uma sintaxe simples e intuitiva, permitindo ajustar a quebra de palavras de acordo com a necessidade. Veja as classes disponíveis:

Classes Utilitárias	Comportamento
`w-nobreak`	*Usa o **comportamento padrão** de quebra de palavras, onde o texto só é quebrado em **espaços em branco** ou **pontos naturais**. Palavras longas podem ultrapassar os limites do contêiner se não houver espaços suficientes.*
`w-break`	*Permite que palavras longas sejam quebradas em pontos arbitrários para se ajustar ao contêiner, evitando overflow. Ideal para textos em idiomas ocidentais com palavras extensas.*
`w-break-all`	*Quebra o texto em qualquer ponto, inclusive dentro de palavras. Útil para idiomas sem espaços, como chinês ou japonês.*
`w-break-keep-all`	*Impede a quebra dentro de palavras, mantendo-as intactas. Pode causar overflow se as palavras forem longas, sendo ideal para preservar a integridade de palavras em idiomas com espaços.*

Como Usar

Aplique essas classes a elementos de texto como `<p>`, `<div>` ou `<span>`. Elas são especialmente úteis em layouts responsivos ou para textos multilíngues com diferentes necessidades de quebra.

Exemplo 1: Comportamento Padrão

```
<div class="w-300px w-nobreak">
  <p>Palavra longa: Supercalifragilisticexpialidocious</p>
  <!-- A palavra longa pode ultrapassar o contêiner de 300px, pois a quebra só ocorre nos espaços. -->
</div>
```

Exemplo 2: Quebra de Palavras Longas

```
<div class="w-300px w-break">
  <p>Palavra longa: Supercalifragilisticexpialidocious</p>
  <!-- A palavra é quebrada para se ajustar ao contêiner, evitando overflow. -->
</div>
```

Exemplo 3: Quebra em Qualquer Ponto

```
<div class="w-300px w-breakall">
  <p>Texto em chinês: 这是一个很长的句子，没有空格。</p>
  <!-- O texto é quebrado em qualquer ponto, ajustando-se ao contêiner mesmo sem espaços. -->
</div>
```

Exemplo 4: Preservar Palavras

```
<div class="w-300px w-break-keep-all">
  <p>Texto em japonês: これは非常に長い文で、単語を壊さないようにします。</p>
  <!-- As palavras são mantidas intactas, mas o texto pode ultrapassar o contêiner se for muito longo. -->
</div>
```

Dicas de Uso

- **Idiomas sem Espaços**: Use `w-break-all` para textos em *chinês*, *japonês* ou *tailandês*.
- **Preservar Palavras**: Prefira `w-break-keep-all` para idiomas como *português* ou *inglês*, mantendo a legibilidade.
- **Combinações**: Combine com classes de `overflow` (ex.: `overflow-hidden`) para gerenciar excessos de texto.
- **Responsividade**: Use `w-break` com larguras dinâmicas (ex.: `w-100%`) para layouts adaptáveis.
- **Testes**: Verifique o comportamento em diferentes navegadores, especialmente para idiomas específicos.

Notas

O uso de `w-break-all` pode afetar a legibilidade; **aplique com cuidado** e teste o resultado visual.

As classes `w-nobreak` e `w-break` são mais comuns em **idiomas ocidentais**, enquanto `w-break-all` e `w-break-keep-all` atendem a necessidades de **idiomas asiáticos** ou **textos técnicos**.

Text-Wrap

As classes utilitárias de `text-wrap` no **WPF** permitem controlar como o texto é quebrado e exibido em um elemento, utilizando a propriedade CSS `text-wrap`. Elas são ideais para ajustar a apresentação de texto em layouts variados, desde evitar quebras de linha até aplicar modos avançados de formatação.

A propriedade `text-wrap` define o comportamento de quebra de linha do texto dentro de um contêiner. Com as classes do WPF, você pode facilmente aplicar esses comportamentos sem escrever CSS manualmente, garantindo consistência e praticidade no design.

Classes Disponíveis

- `t-nowrap`: Impede que o texto quebre em várias linhas, exibindo-o em uma única linha contínua. Útil para títulos ou textos que devem permanecer compactos, mas pode gerar overflow se o conteúdo for maior que o contêiner.
- `t-wrap`: Permite que o texto quebre em linhas de forma natural, conforme o espaço disponível e as regras de quebra de palavras do navegador.
- `t-wrap-auto`: Deixa o navegador escolher o modo de quebra mais adequado.
- `t-wrap-balance`: Distribui o texto de forma equilibrada entre as linhas, ideal para parágrafos curtos.
- `t-wrap-stable`: Garante consistência no layout durante a renderização, evitando mudanças bruscas.
- `t-wrap-pretty`: Prioriza a legibilidade, ajustando as quebras para um fluxo visual mais agradável.

Exemplos de Uso

Exemplo 1: Texto em Linha Única

```
<span class="t-nowrap">Este texto longo não será quebrado em várias linhas.</span>
```

Exemplo 2: Quebra de Linha Padrão

```
<p class="t-wrap">Este texto será quebrado em linhas conforme o espaço disponível no contêiner.</p>
```

Exemplo 3: Modos Específicos

```
<p class="t-wrap-balance">Este parágrafo será balanceado para linhas de tamanhos similares.</p>
<p class="t-wrap-pretty">Este texto é ajustado para melhor legibilidade e fluidez.</p>
```

Dicas e Notas

> **Overflow**: Use `t-nowrap` com classes de overflow (ex.: `overflow-hidden`) para gerenciar texto que ultrapassa o contêiner.

> **Compatibilidade**: Modos como *balance* e *pretty* podem não ser suportados em navegadores antigos. Teste em seu ambiente alvo.

Word-Wrap

As classes utilitárias de `word-wrap` no **WPF** são usadas para controlar o comportamento de quebra de palavras em elementos HTML, aplicando a propriedade CSS `word-wrap`. Elas ajudam a gerenciar textos longos em contêineres com largura limitada.

Classes Disponíveis

- `w-nowrap`: Impede a quebra de palavras, mantendo o texto em uma única linha. Se o texto for muito longo, ele pode ultrapassar os limites do contêiner, **resultando em overflow**.
- `w-wrap`: Permite que palavras longas sejam quebradas em pontos arbitrários, ajustando o texto ao tamanho do contêiner e **evitando overflow**.

Exemplos de Uso

Exemplo 1: Impedir Quebra de Palavras

```
<div class="w-300px w-nowrap">
  <p>Supercalifragilisticexpialidocious</p>
  <!-- A palavra longa não será quebrada e pode exceder a largura de 300px do contêiner. -->
</div>
```

Exemplo 2: Permitir Quebra de Palavras

```
<div class="w-300px w-wrap">
  <p>Supercalifragilisticexpialidocious</p>
  <!-- A palavra será quebrada para se ajustar ao contêiner de 300px. -->
</div>
```

Line-Height

As classes utilitárias de `line-height` no **WPF** permitem ajustar a altura das linhas de texto de forma rápida e consistente. Elas controlam o espaçamento entre linhas, melhorando a legibilidade e a estética do texto em parágrafos, títulos e outros elementos.

Como Funcionam as Classes de Line Height

As classes seguem o padrão **lh-[valor][unidade]**, onde:

- **[valor]** é um número (*inteiro* ou *decimal*) que **define o tamanho da altura da linha**.
- **[unidade]** é uma unidade de medida obrigatória: `px`, `rem`, `em` ou `%`.

Regras CSS Geradas

Cada classe é convertida para a propriedade CSS `line-height`. Por exemplo:

- `lh-20px` → `line-height: 20px`;
- `lh-1.5rem` → `line-height: 1.5rem`;
- `lh-150%` → `line-height: 150%`;

Como Usar

Aplique as classes diretamente aos elementos HTML, como `<p>`, `<h1>`, `<li>`, etc., para ajustar o espaçamento entre linhas.

Exemplo 1: Texto de Corpo

```
<p class="lh-1.6rem">
  <!-- Este parágrafo tem uma altura de linha de 1.6rem, ideal para legibilidade em textos longos. -->
</p>
```

Exemplo 2: Títulos

```
<h2 class="lh-1.2em">
  <!-- Este título usa uma altura de linha mais compacta, perfeita para textos curtos. -->
</h2>
```

Notas

> Essas classes exigem uma unidade de medida (`px`, `rem`, `em`, ou `%`). Valores sem unidade (ex.: `lh-1.5`) não são suportados por este conjunto de utilitários.

Font-Size

As classes utilitárias de `font-size` no **WPF** permitem ajustar o tamanho da fonte de elementos HTML de maneira rápida e consistente. Elas utilizam a propriedade CSS `font-size` e suportam unidades absolutas e relativas, como `px`, `em`, `rem` e `%`, oferecendo flexibilidade para diferentes contextos de design.

Como Funcionam as Classes de Font-Size

As classes seguem o padrão **fs-[valor][unidade]**, onde:

- **[valor]** é um número (*inteiro* ou *decimal*) que **define o tamanho da fonte**.
- **[unidade]** é uma unidade de medida obrigatória: `px`, `em`, `rem` ou `%`.

Regras CSS Geradas

Cada classe é traduzida diretamente para a propriedade CSS `font-size`. Por exemplo:

- `fs-16px` → `font-size: 16px`;
- `fs-1.5rem` → `font-size: 1.5rem`;
- `fs-120%` → `font-size: 120%`;

Como Usar

Aplique as classes diretamente a elementos de texto, como `<p>`, `<h1>`, `<span>`, etc., para ajustar o tamanho da fonte. As classes são ideais para criar hierarquias tipográficas claras e consistentes.

Exemplo 1: Texto de Corpo

```
<p class="fs-16px">Este parágrafo tem fonte de 16px.</p>
<p class="fs-1.2rem">Este parágrafo tem fonte de 1.2rem, relativo ao elemento raiz.</p>
```

Exemplo 2: Títulos

```
<h1 class="fs-32px">Título Principal (32px)</h1>
<h2 class="fs-24px">Subtítulo (24px)</h2>
```

Exemplo 3: Tamanhos Relativos

```
<div class="fs-1.5em">
  <p>Este texto é 1.5 vezes maior que o pai.</p>
  <p class="fs-80%">Este texto é 80% do tamanho do pai (1.5em * 0.8 = 1.2em do original).</p>
</div>
```

Exemplo 4: Percentuais

```
<p class="fs-150%">Este texto é 150% maior que o tamanho da fonte do pai.</p>
```

Dicas e Melhores Práticas

> **Hierarquia**: Combine diferentes tamanhos para criar uma hierarquia visual clara entre títulos, subtítulos e textos de corpo. Mantenha-se consistente em quais tags html utilizar para quais tamanhos;

> **Acessibilidade**: Evite tamanhos muito pequenos (ex.: `fs-10px`) para garantir legibilidade, especialmente em dispositivos móveis. O *tamanho recomendado para texto é a partir de* **12 pixels de tamanho.**

> **Testes**: Verifique o comportamento em diferentes navegadores e tamanhos de tela para garantir que o texto se ajuste corretamente.

Text-Decoration

As classes utilitárias de `text-decoration` no **WPF** permitem adicionar ou remover decorações de texto, como *sublinhados*, *linhas sobre o texto* ou *tachados*, de **forma rápida e consistente**. Elas utilizam a propriedade CSS `text-decoration` e oferecem uma variedade de estilos para personalizar a aparência do texto.

Como Funcionam as Classes de Text-Decoration

As classes seguem o padrão **t-[estilo]**, onde **[estilo]** é um dos valores permitidos: `underline`, `dashed`, `double`, `line-through`, `overline`, `solid`, `wavy` ou `none`.

Classes Utilitárias	Comportamento
`t-underline`	*Adiciona um **sublinhado sólido** ao texto.*
`t-dashed`	*Adiciona um **sublinhado tracejado** ao texto.*
`t-double`	*Adiciona um **sublinhado duplo** ao texto.*
`t-line-through`	*Adiciona um **tachado** ao texto.*
`t-overline`	*Adiciona uma **linha sobre** o texto.*
`t-solid`	*Adiciona um **sublinhado sólido** (similar a `underline`, mas pode ser usado para consistência).*
`t-wavy`	*Adiciona um **sublinhado ondulado** ao texto.*
`t-none`	***Remove qualquer decoração** de texto.*

Regras CSS Geradas

Cada classe é traduzida diretamente para a propriedade CSS `text-decoration`, seguindo o template **.t-[estilo] { text-decoration: [estilo]; }**. Por exemplo:

- `t-underline` → `text-decoration: underline`;
- `t-dashed` → `text-decoration: dashed`;
- `t-none` → `text-decoration: none`;

Exemplos de Uso

Exemplo 1: Sublinhado Simples

```
<p class="t-underline">Este texto está sublinhado.</p>
```

Exemplo 2: Tachado

```
<p class="t-line-through">Este texto está tachado.</p>
```

Exemplo 3: Sublinhado Ondulado

```
<p class="t-wavy">Este texto tem um sublinhado ondulado.</p>
```

Exemplo 4: Remover Decoração

```
<a href="#" class="t-none">Este link não tem sublinhado.</a>
```

Guia de Boas Práticas

> **Acessibilidade**: Use `t-line-through` para indicar **texto excluído** ou **desatualizado**, mas certifique-se de que o significado seja claro para todos os usuários.

> **Consistência**: Use `t-none` para remover sublinhados de links ou outros elementos que por padrão têm decorações.

> **Navegadores**: Alguns estilos, como `dashed` ou `wavy`, podem ter renderizações diferentes em navegadores antigos. Teste em seu ambiente alvo.

Notas

As classes de `text-decoration` são aplicadas diretamente ao elemento e afetam todo o texto dentro dele.

Para decorações mais complexas, como múltiplas linhas ou cores específicas, considere usar CSS personalizado.

O valor `solid` é similar a `underline`, mas pode ser útil para manter a consistência na nomenclatura.

Posicionamento

Aprenda mais sobre como utilizar classes relacionadas à *position, overflow, z-index, box-sizing* entre outros que regem a posição espacial do seu elemento visual dentro do ambiente de trabalho.

Posicionamento

Essas classes oferecem diferentes opções de posicionamento para elementos HTML. Elas são úteis ao precisar controlar a posição de um elemento em relação ao seu contêiner ou à janela do navegador.

Classes Disponíveis

- `.static`: Define o posicionamento do elemento como estático, ou seja, ele segue o fluxo normal do documento e não é afetado por outros elementos.
- `.sticky`: Define o posicionamento do elemento como "sticky", o que significa que ele se comporta como "relative" até que atinja um determinado ponto de rolagem. Após atingir esse ponto, ele se torna "fixed", mantendo-se fixo na tela.
- `.relative`: Define o posicionamento do elemento como relativo ao seu posicionamento normal. Isso permite que você mova o elemento usando as propriedades **`top`**, **`right`**, **`bottom`** e **`left`**.
- `.absolute`: Define o posicionamento do elemento como absoluto em relação ao seu ancestral mais próximo que possui uma posição diferente de `static`. Se nenhum ancestral for encontrado, o elemento será posicionado em relação ao elemento `<html>`.
- `.fixed`: Define o posicionamento do elemento como fixo em relação à janela do navegador. Ele permanecerá na mesma posição, independentemente da rolagem da página.

Uso

Para aplicar esses estilos de posicionamento a um elemento HTML, basta adicionar a classe correspondente conforme necessário. Por exemplo:

```
<div class="static">
  Este é um elemento com posicionamento estático.
</div>

<div class="sticky">
  Este é um elemento com posicionamento "sticky".
</div>

<div class="relative">
  Este é um elemento com posicionamento relativo.
```

```
</div>

<div class="absolute">
    Este é um elemento com posicionamento absoluto.
</div>

<div class="fixed">
    Este é um elemento com posicionamento fixo.
</div>
```

Cada classe de posicionamento alterará o comportamento do elemento conforme descrito acima, permitindo um controle preciso sobre o layout e a aparência da página.

Configurando posicionamento por funções

Há casos onde será necessário instruir o elemento a se comportar de uma maneira ainda mais específica e, portanto, o WPF oferece algumas funções que auxiliam o usuário a configurar o posicionamento do seu elemento de forma específica diretamente no seu arquivo SCSS/SASS:

Confira elas aqui:

setPosition()

centerPosition()

Esses mixins oferecem maneiras simples e flexíveis de configurar o posicionamento de elementos em seus projetos, permitindo uma maior personalização e controle sobre o layout.

Além dos mixins, você também pode optar por usar uma versão dessas funções *em classe.* Juntamente com as classes auxiliares que definem um posicionamento. Se elas forem `absolute`, `relative` ou `fixed` , é possível utilizar as funções `top()`, `right()`, `bottom()` e `left()`, com valores em **%, px, dvh** ou **dvw** para setar valores **diretamente no atributo class** do seu componente HTML. É possível até mesmo utilizar o mixin centerPosition() via classe com a função `center()`.

Transbordamento

As classes de overflow são utilizadas para definir o comportamento de overflow (transbordamento) de conteúdo para um elemento. Elas oferecem opções para controlar o comportamento de overflow em várias direções.

Uso Geral:

```
<div class="overflow-y">
   <!-- Conteúdo aqui -->
</div>
```

Opções Disponíveis:

- **`.overflow`**:
  - Aplica **`overflow: auto !important;`** para habilitar barras de rolagem quando necessário.
- **`.overflow-hidden`**:
  - Aplica **`overflow: hidden !important;`** para ocultar o conteúdo que ultrapassa os limites do elemento.
- **`.overflow-scroll`**:
  - Aplica **`overflow: scroll !important;`** para exibir barras de rolagem, independentemente se o conteúdo ultrapassa os limites ou não.
- **`.overflow-visible`**:
  - Aplica **`overflow: visible !important;`** para exibir todo o conteúdo, mesmo que ele ultrapasse os limites do elemento.

Opções de Direção:

As classes também oferecem opções para controlar o comportamento de overflow em direções específicas: vertical e horizontal.

- **`.overflow-y`**:
  - Aplica **`overflow-y: auto !important;`** para habilitar barras de rolagem vertical quando necessário.
- **`.overflow-y-hidden`**:

    - Aplica `overflow-y: hidden !important;` para ocultar o conteúdo que ultrapassa os limites verticalmente.
- `.overflow-y-scroll`:
    - Aplica `overflow-y: scroll !important;` para exibir barras de rolagem vertical, independentemente se o conteúdo ultrapassa os limites ou não.
- `.overflow-y-visible`:
    - Aplica `overflow-y: visible !important;` para exibir todo o conteúdo verticalmente, mesmo que ele ultrapasse os limites do elemento.
- `.overflow-x`:
    - Aplica `overflow-x: auto !important;` para habilitar barras de rolagem horizontal quando necessário.
- `.overflow-x-hidden`:
    - Aplica `overflow-x: hidden !important;` para ocultar o conteúdo que ultrapassa os limites horizontalmente.
- `.overflow-x-scroll`:
    - Aplica `overflow-x: scroll !important;` para exibir barras de rolagem horizontal, independentemente se o conteúdo ultrapassa os limites ou não.
- `.overflow-x-visible`:
    - Aplica `overflow-x: visible !important;` para exibir todo o conteúdo horizontalmente, mesmo que ele ultrapasse os limites do elemento.