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
- Atribuições de Responsividade, Importância e Pseudo Elementos
- Declarando e Usando Variáveis CSS

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.