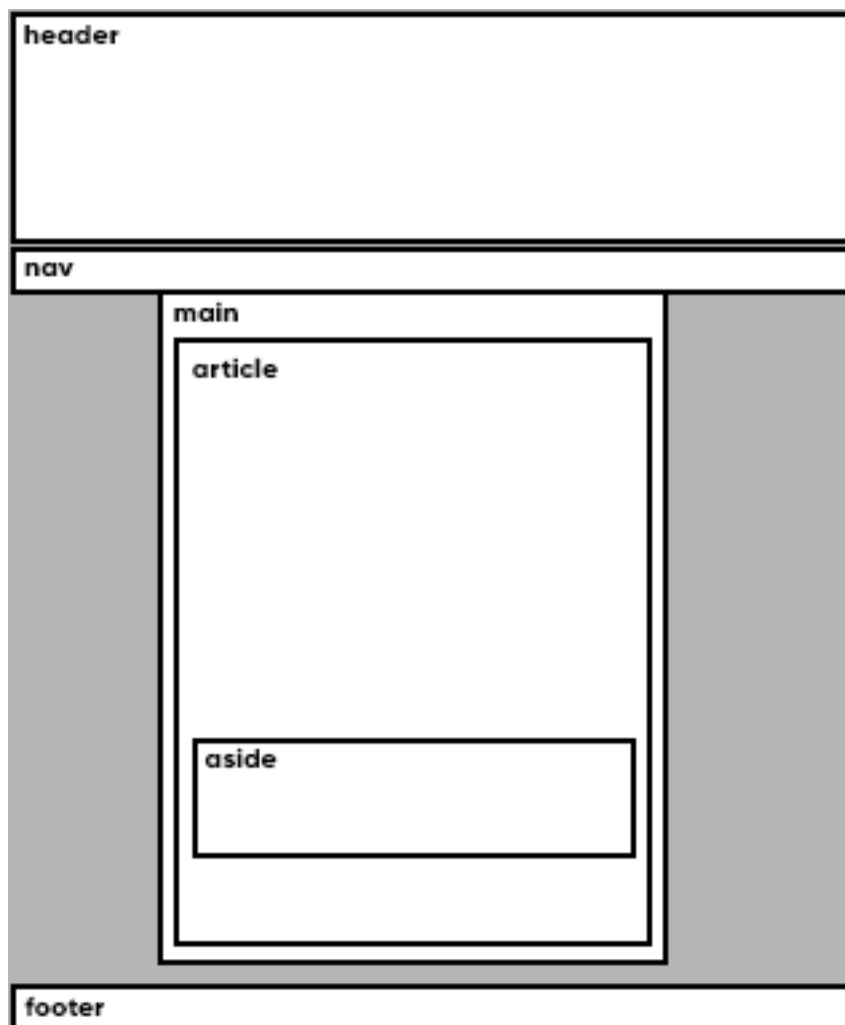


# A ideia do projeto

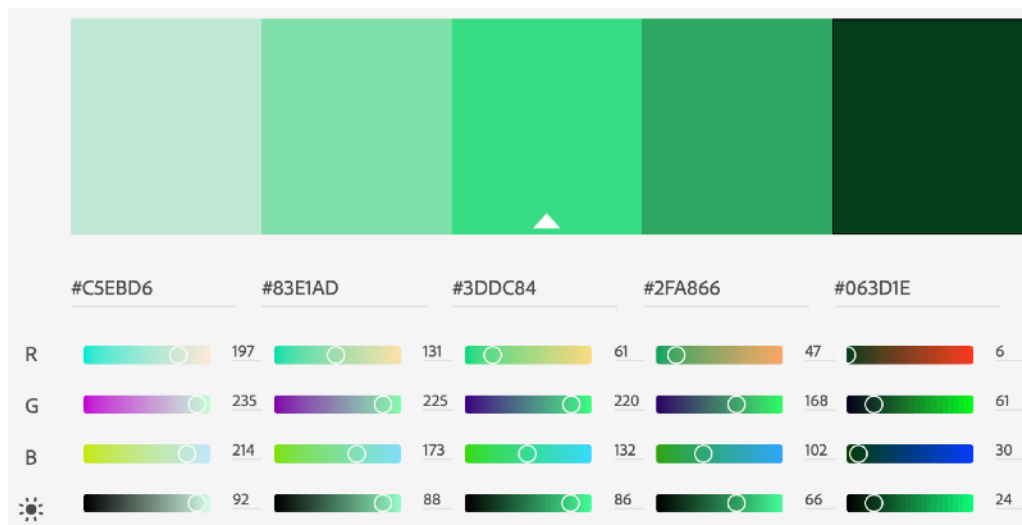
A ideia inicial é criar uma espécie de site de notícias, só que com uma notícia só 😊. O objetivo aqui é apenas ensinar como organizar o conteúdo e apresentá-lo em forma de página Web.

Tudo começa desenhando o que chamamos de *wireframe*, planejando qual vai ser a estrutura e comportamento do site. Ele vai servir de base na hora de planejar as caixas que farão parte da página. Para esse nosso primeiro mini-projeto, planejei o seguinte *layout*:



**CRIE SEUS WIREFRAMES:** Existe uma ferramenta muito legal para planejar seus *layouts*, tanto para sites quanto para aplicativos desktop e até apps de celular: o **MockFlow**. Como podemos imaginar, existem limitações na versão *free*, mas dá pra imaginar como o site vai ficar antes de começar a mexer no código.

Em seguida, vamos decidir a paleta de cores que será utilizada. Optei por uma paleta baseada em **monocromia** a partir de tons de verde (já que vamos falar sobre Android), mas você pode usar a técnica que quiser para decidir suas cores básicas que serão usadas. Usei o **Adobe Color** (que aprendemos a usar no capítulo 13) e optei pelas seguintes cores:



Por fim, vamos falar das fontes escolhidas. A primeira, **Bebas Neue**, será a fonte para os destaques principais. Já que vamos fazer uma matéria sobre o mundo Android, optei por uma fonte chamada **iDroid**. Já para o texto padrão, escolhi a **Arial**.

**LOREM IPSUM**

**Bebas Neue**, cursive

**LOREM IPSUM**

**iDroid**, sans-serif

Lorem, ipsum dolor sit amet consectetur  
adipiscing elit. Eum, quo temporibus  
voluptates mollitia eius voluptas qui,  
officia blanditiis voluptatum ipsum  
exercitationem incidunt. Nemo aperiam  
laboriosam corporis magnam aliquid  
perspiciatis quae?

**Arial**, Helvetica, sans-serif

# Já ouviu falar em responsividade?



Ainda que tenha optado por criar um projeto simples, não vamos abrir mão da versatilidade, pois queremos que nosso site possa ser visualizado em vários dispositivos com tamanhos diferentes de tela. O nome que damos a essa característica é **responsividade**. Um site dito responsivo vai ser capaz adaptar os tamanhos dos seus componentes para manter o site plenamente visível em qualquer tela. É claro que por enquanto, não aprendemos muita coisa para permitir essa adaptação completamente para todo tipo de situação, mas com o que vimos até aqui já dá pra começar.

## Largura e altura adaptáveis

No capítulo anterior, vimos as propriedades para largura (`width`) e altura (`height`) de uma caixa. Se não configurarmos a largura, por exemplo, ela vai ocupar 100% do seu contêiner. Porém, para que nosso site se adapte mais facilmente ao tamanho da tela, uma das técnicas básicas que podem ser utilizadas é deixar algumas caixas estrategicamente um pouco mais flexíveis indicando valores mínimos (`min-width` / `min-height`) e máximos (`max-width` / `max-height`). Para o nosso projeto, vamos definir os limites da largura da caixa `<main>` onde estará contido o conteúdo principal do artigo.

```
min-width: 400px;  
max-width: 800px;
```

A configuração acima vai garantir que a caixa em questão não fique com largura menor que 400px (para telas pequenas) e nem seja maior que 800px (para telas muito grandes).

Para medidas entre 400px e 800px, o tamanho será adaptável e a caixa vai ocupar 100% da largura do contêiner. Isso vai facilitar bastante a adaptação do conteúdo a diversos tipos de tela.

**DÁ PRA SER MAIS ADAPTÁVEL?** É possível aplicar outras técnicas adicionais para dar ainda mais capacidade de adaptação do conteúdo, como as *media queries* com a regra `@media` e as caixas flexíveis com as *flex boxes*. Mais pra frente, abordaremos esses conteúdos no curso, mas por enquanto as medidas adaptáveis já vão quebrar um galho.

# IMPORTANTE: variáveis em CSS

Com essa dica, você vai levar suas folhas de estilo a um outro nível! Vamos aprender a utilizar variáveis personalizadas com CSS para cadastrar os esquemas de cores e fontes do nosso site e isso vai facilitar muito na hora de efetuar eventuais mudanças estéticas no nosso site.

Para criarmos variáveis para nossas configurações, devemos definir uma área de definições dentro do seu estilo para uma pseudo-classe chamada `:root`, que definem as configurações para a raiz de uma árvore, que vai servir para o documento inteiro.

Note que, pelas declarações criadas ao lado, definimos nove variáveis com as configurações de cores e fontes que definimos no início desse capítulo.

```
:root {  
  --cor0: #c5ebd6;  
  --cor1: #83E1AD;  
  --cor2: #3DDC84;  
  --cor3: #2FA866;  
  --cor4: #1A5C37;  
  --cor5: #063d1e;  
  --fonte0: Arial, Helvetica, sans-serif;  
  --fonte1: 'Bebas Neue', cursive;  
  --fonte2: 'Android', sans-serif;  
}
```

A partir de agora, definir cores e fontes em nossos elementos HTML ficará extremamente mais fácil e personalizável, utilizando a função `var()`.

```
body {  
  font-family: var(--fonte0);  
  color: var(--cor0);  
  background-color: var(--cor1);  
}
```

**IMPORTANTE:** As variáveis personalizadas em CSS devem ter seus nomes iniciando com dois traços obrigatoriamente.

Olhando as declarações feitas para o seletor de `body` acima, pode parecer inicialmente que elas ficaram um pouco mais confusas. Mas imagine que seu cliente mude as definições de cores e fontes. Quantas alterações teriam que ser feitas para deixar tudo em dia com as novas especificações? Ao usar variáveis, tudo se simplifica muito, pois basta atualizar as variáveis definidas em `:root`.

# Uso do seletor \* em CSS

Existe também um seletor especial das CSS que é o asterisco (\*), ele tem uma função muito especial, pois basicamente ele aplica uma configuração padrão para **TODOS** os elementos do código HTML ao qual o estilo está sendo aplicado.

No nosso caso, para o nosso mini-projeto, nós vamos utilizar esse seletor global para eliminar as eventuais margens que os navegadores (*user agents*) adicionam a alguns elementos. Isso vai facilitar bastante, pois vai permitir personalizar as medidas que vão aparecer na tela para cada elemento individualmente.

## Espaçamento entrelinhas em Textos

Outra configuração muito importante que podemos fazer para textos muito longos é o de espaçamento entre as linhas do nosso texto. Usando a propriedade `line-height`, podemos dizer qual é o tamanho do espaço entre uma linha e a outra do texto. O valor padrão na maioria dos navegadores é algo próximo ao 1.2em, mas veja a seguir a aplicação de outros valores.

Provavelmente você sabe que o sistema operacional <b>Android</b> , mantido pelo <b>Google</b> é um dos mais utilizados para dispositivos móveis em todo o mundo. Mas tavez você não saiba que o seu simpático mascote tem um nome e uma história muito curiosa? Pois acompanhe esse artigo para aprender muita coisa sobre esse robozinho.	Provavelmente você sabe que o sistema operacional <b>Android</b> , mantido pelo <b>Google</b> é um dos mais utilizados para dispositivos móveis em todo o mundo. Mas tavez você não saiba que o seu simpático mascote tem um nome e uma história muito curiosa? Pois acompanhe esse artigo para aprender muita coisa sobre esse robozinho.	Provavelmente você sabe que o sistema operacional <b>Android</b> , mantido pelo <b>Google</b> é um dos mais utilizados para dispositivos móveis em todo o mundo. Mas tavez você não saiba que o seu simpático mascote tem um nome e uma história muito curiosa? Pois acompanhe esse artigo para aprender muita coisa sobre esse robozinho.
<code>line-height: 1.2em;</code>	<code>line-height: 1.5em;</code>	<code>line-height: 2.0em;</code>



## Sombras em Textos

No capítulo anterior, falamos sobre as sombras em caixas, utilizando a propriedade `box-shadow`. Pois saiba que também existe uma propriedade específica para criar sombras em textos: o `text-shadow`.

A propriedade `text-shadow` também pode ter quatro parâmetros principais:

1. **Deslocamento horizontal** (*h-offset*): quanto a sombra vai andar para o lado direito (valores negativos causam deslocamento para a esquerda)
2. **Deslocamento vertical** (*v-offset*): quanto a sombra vai andar para baixo (valores negativos causam deslocamento para cima)
3. **Embaçamento** (*blur*): quanto a sombra vai se espalhar pelo fundo
4. **Cor** (*color*): cor da sombra. É possível usar transparência.

```
text-shadow: 2px 2px 0px ■ rgba(0, 0, 0, 0.466);
```

	
Sem text-shadow	Com text-shadow

Como você deve ter percebido olhando as imagens acima, a sombra em textos serve para destacar a letra e seu fundo, criando um contraste entre elas.

## Personalizando ainda mais as listas

No **capítulo 9** nós aprendemos a criar listas de vários tipos. Agora, vou te mostrar como criar mais personalizações e a criar um ótimo resultado visual.



No nosso projeto, quero adicionar uma lista com todas as 14 versões principais do sistema Android. Se fizermos usando apenas os elementos comuns sem configurá-los, teremos uma listagem que ocupa um grande espaço vertical. A solução aqui é dividir a lista em duas colunas e modificar o marcador para personalizar ainda mais a exibição do conteúdo.

```
ul {  
  list-style-type: '\2714\0020\0020';  
  columns: 2;  
  list-style-position: inside;  
}
```

A primeira linha de declarações faz com que o marcador seja personalizado com o parâmetro `list-style-type`. O valor `\2714` corresponde ao símbolo ✓ que tem o código Unicode U+2714 (confira no site da Emojipedia). O valor `\0020` corresponde a um espaço em branco (também pode ser `\00A0`).

A segunda declaração vai organizar a lista em duas colunas. O total de elementos da lista com `<li>` será dividido em duas partes iguais (ou quase) e o resultado será colunado.

Por fim, a última declaração vai fazer com que os marcadores sejam exibidos na parte interna da caixa que contém a lista. Analise as imagens abaixo e perceba que, por padrão, a caixa de uma lista não inclui os marcadores. Alteramos essa característica usando a declaração `list-style-position` com o valor `inside`, já que a lista vai estar dentro de um `<aside>` no nosso documento HTML.

	
<code>list-style-position: outside;</code>	<code>list-style-position: inside;</code>

## Vídeos do YouTube mais flexíveis

Aprendemos no **capítulo 11** como deixar as imagens mais “flexíveis” usando a tag `<picture>`. Também aprendemos a adicionar vídeos usando várias técnicas, inclusive aqueles que estão hospedados em serviços especializados como **Vimeo** e **YouTube**.

O problema é que, quando incorporamos um vídeo do YouTube ou Vimeo, isso é feito através de uma tag `<iframe>` que já vem com as configurações de um tamanho fixo e precisamos alterar isso para que nossa interface possa se tornar mais responsiva e adaptar o tamanho do vídeo dinamicamente, principalmente para telas pequenas.



Para isso, vamos colocar o `<iframe>` de incorporação dentro de uma `<div>` para que o vídeo esteja limitado dentro de um contêiner.

```
<div class="video">
  <!-- AQUI VAI O CÓDIGO DO IFRAME -->
</div>
```

A partir daí, vamos fazer configurações de estilo para os dois elementos:

```
div.video {
  position: relative;
  background-color: var(--cor4);
  height: 0px;
  margin-left: -20px;
  margin-right: -20px;
  margin-bottom: 15px;
  padding-bottom: 59%;
}

div.video > iframe {
  position: absolute;
  top: 5%;
  left: 5%;
  width: 90%;
  height: 90%;
}
```

O que nos importa mais aqui é entender o funcionamento da propriedade `position`, que é a única que não vimos até aqui. O valor padrão para essa propriedade de posicionamento é `static`, que mantém a hierarquia conforme estabelecido no documento HTML.

Na `div`, nós colocamos o valor como `relative` para que seja considerado o posicionamento atual do elemento de divisão e que ele se mantenha adaptável para o caso de alteração no tamanho do navegador.

Já dentro do `iframe`, nós usamos o posicionamento `absolute` para que a `div` - que é o seu contêiner - torne-se o ponto de partida para o posicionamento do frame. A partir daí, podemos utilizar propriedades para configurar o deslocamento à esquerda (`left`) e ao topo (`top`) e seu tamanho em largura (`width`) e altura (`height`), todos em porcentagem de tela.