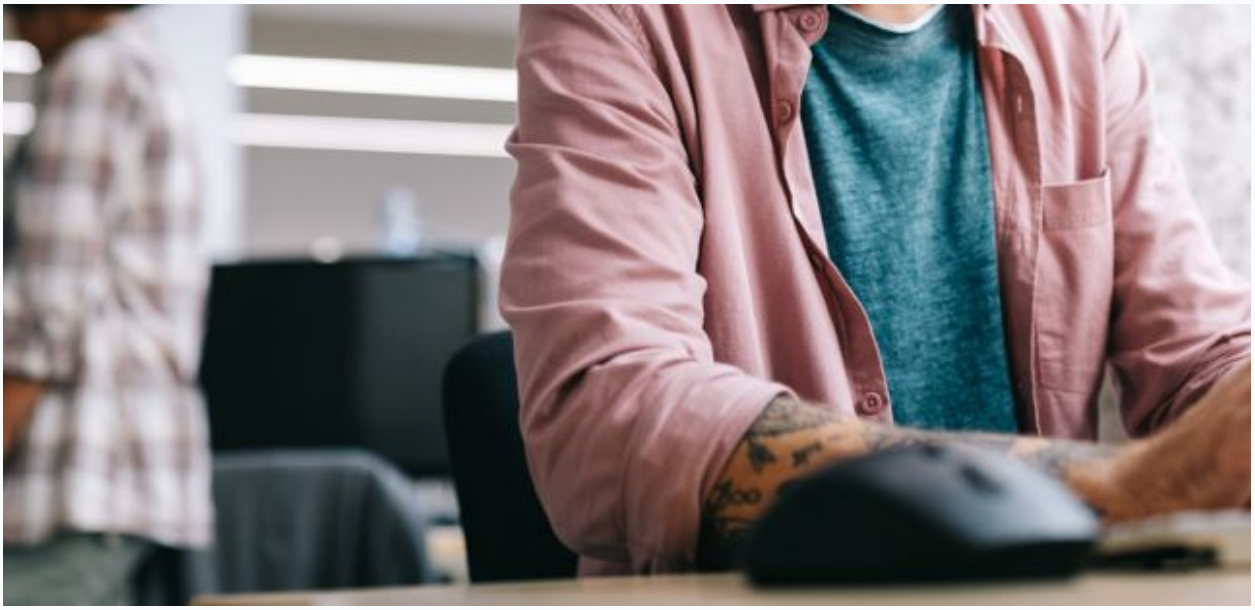
Criando Layouts com CSS Grid Layout



M

matheus-castiglioni

Atualizado em 08/10/2021

Esse artigo faz parte da
Formação Front-end

Um recurso que nos permite "dizer" aos elementos onde queremos que eles fiquem, qual vai ser o tamanho deles, quanto de espaçamento vai existir, criar "lacunas" ou "buracos" que serão preenchidos por nossas tags HTML. Com CSS Grid Layout é exatamente isso que conseguimos.

Imagine que um amigo, chegou em você e disse:

> *Eu preciso criar um layout com um cabeçalho, um menu lateral, o conteúdo principal e um rodapé*

Se você está acostumado a criar páginas HTML5 com CSS3 já deve ter imaginado toda a estrutura, algo parecido com:

```
<header>Cabeçalho</header>
<aside>Menu lateral</aside>
<main>Conteúdo principal</main>
<footer>Rodapé</footer>
```

Até ai tudo bem, certo? Certo. Montar a estrutura das tags é a parte mais fácil... Porém,agora precisamos posiciona-las da seguinte maneira:

- Header: Em cima da página
- Aside: Na parte esquerda da página
- Main: Na parte direita da página
- Footer: Em baixo da página

Com isso iremos ter a seguinte estrutura:

Para você que já esta pensando nos códigos e está ancioso para começar a escrever seu `.css`, acredito que deva estar imaginando diversas maneiras de chegar ao design que nosso amigo esta precisando:

> *Posso usar flexbox, talvez eu use float, se der na louca eu utilizo o position, etc...*

Pois é, normalmente eu faria esse design com `float` ou `flexbox`, medindo o tamanho de cada elemento, adicionando um pouco de `margin` e `padding`... Mas, para esse post, iremos utilizar a nova *feature* do CSS chamada "**CSS Grid Layout**".

Conhecendo o CSS Grid Layout

Como criar layouts era uma tarefa comum no dia-a-dia e às vezes um tanto quanto chata de se implementar, a galera pensou: "Será que não seria possível melhorar esse processo, deixando mais rápido e eficiente?", foi nesse contexto que criou-se a nova funcionalidade chamada de **CSS Grid Layout**.

Começando com nosso código

Sem mais delongas, vamos começar a codificar e pôr em prática tudo isso. A primeira parte será definir o **HTML** da página:

```
<header class="o-header">Header</header>
<aside class="o-aside">Aside</aside>
<main class="o-footer">Main</main>
<footer class="o-footer">Footer</footer>
```

Com isso, iremos ter o seguinte resultado:

Edit in JSFiddle

- Result
- HTML

Header
Aside
Main
Footer

Como podemos ver, tudo está sem posicionamento e espaços.

Nosso primeiro passo será definir qual tag vai receber as "lacunas" para posteriormente adicionarmos nossos elementos. Bom, se tudo esta dentro de `body`, temos um bom candidato:

```
body {
  display: grid;
}
```

Vejam que o `display` ganhou um novo valor chamado `grid`. Com isso estamos dizendo: "Olha, navegador, a tag `body` vai receber um grid, assim posso informar onde cada tag vai ficar". Mas, afinal, onde estamos dizendo e informando o posicionamento de cada tag ?

Posicionando nossas tags

Para posicionar nossas tags, devemos praticamente "desenhar" no CSS, onde elas estarão:

```
body {
  display: grid;
  grid-template-areas:
    "header header header"
    "aside main main"
    "footer footer footer";
}
```

Podemos ver o resultado:

Edit in JSFiddle

- Result
- HTML
- CSS

Header Aside Main
Footer

> *Pera aí, Matheus, que maluquisse é essa? O que é "header header header"?*

Calma, vamos por partes, vou explicar as linhas que foram adicionadas.

Entendendo o Template Areas

Com a propriedade `css-template-areas` "desenhamos" nosso *layout*, assim conseguimos informar como e por quais elementos ele vai ser composto:

- "header header header": Aqui estamos dizendo que a primeira linha vai ser composta por um *header*.
- "aside main main": Aqui estamos dizendo que a segunda linha vai ser composta por um *aside* na esquerda e por um *main* na esquerda.
- "footer footer footer": Aqui estamos dizendo que a terceira e última linha vai ser composta por um *footer*.

Assim conseguimos chegar no *layout* que nosso amigo estava precisando.

Mas por que você declarou o header e o footer três vezes ?

O *header* e *footer* foram declarados três vezes, porque estamos trabalhado com um layout de três colunas, isso deve-se ao fato da segunda linha do nosso *template*.

E por que três colunas ?

Sim, nossa segunda linha poderia ser com duas colunas "aside main" dessa forma, mas então por que repetimos o *main* duas vezes ? Repare no layout que precisamos chegar:

Veja que o *main* é um pouco maior do que o *aside*. Mais precisamente, o *main* é **duas vezes**

maior que o *aside*, por isso, tivemos que repetir o mesmo duas vezes, assim estamos dizendo para o *template*: "Olha template, aqui na segunda linha, vai ter um aside à esquerda e um main à direita, porém, o main deve ser duas vezes maior que o aside".

Resultando em nosso *layout* de três colunas: "aside main main", então por isso, na primeira e terceira linha devemos também ter três colunas.

Legal, agora eu entendi e tirei minhas dúvidas. Vamos então dizer aos elementos que eles devem ficar naquelas "lacunas".

Dizendo aos elementos onde devem ficar

Para dizer aos elementos onde eles devem ficar, ou seja, qual "lacuna" é de cada um, precisamos ir em um por um deles e informar:

```
.o-header {
grid-area: header;
}

.o-aside {
grid-area: aside;
}

.o-main {
grid-area: main;
}

.o-footer {
grid-area: footer;
}
```

Assim, teremos o seguinte resultado:

Edit in JSFiddle

- Result
- HTML
- CSS

Header
Aside Main
Footer

Veja que cada um esta em sua devida posição, repare também que o nome informado na propriedade `grid-area` deve ser **exatamente** igual aos nomes dados no `grid-template-areas`. Seguindo nosso *layout*, esta faltando apenas setar a cor dos elementos, certo ?

```
.o-header, .o-aside, .o-main, .o-footer {
background: #BC20E2;
color: #FDFDFD;
}
```

E, novamente, nosso resultado:

Edit in JSFiddle

- Result
- HTML
- CSS

Mas, espera ai... Nosso *layout* possuí espaçamentos e ocupam a página toda, até agora esta tudo junto e ocupando apenas o tamanho necessário.

Dando espaçamentos entre os elementos

Vamos então, informar o espaço que devem haver entre os elementos:

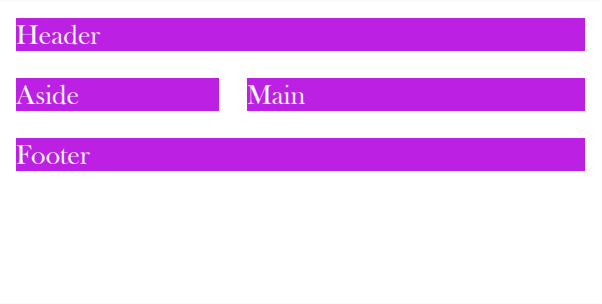
```
body {
grid-gap: 1rem;
```

```
}
```

Com isso, os elementos já devem estar espaçados em `1rem` que seria `16px`:

Edit in JSFiddle

- Result
- HTML
- CSS



Falta apenas eles ocuparem a página toda!

Dizendo o tamanho dos elementos

Para dizer o tamanho que cada *row*(linha) ou *column*(coluna) devem ter, precisamos também dizer ao nosso template:

```
body {
grid-template-columns: auto auto auto;
grid-template-rows: auto 100vh auto;
}
```

Repare novamente que tanto nossas columns ou rows, estão sendo informadas três vezes, isso porque temos um *layout* de três colunas e três linhas, veja também que apenas para a segunda linha declaramos o valor `100vh` que corresponde a altura total da _**viewport**_, as demais linhas serão automáticas.

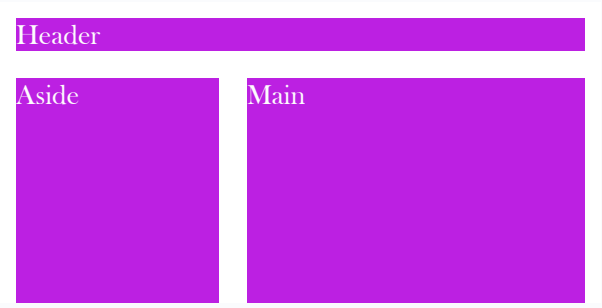
Por padrão, o valor será `auto`, mas apenas descrevi para você saber que pode estar mudando. Para saber quais valores podem ser setados, consulte a documentação: **grid-template-columns** e **grid-template-rows**.

Com isso, devemos ter o seguinte resultado:

Edit in JSFiddle

Edit in JSFiddle

- Result
- HTML
- CSS



Bem próximo relacionado à necessidade de nosso amigo.

Dando os toques finais

Para sanar a necessidade de nosso amigo e entregar o *layout* para ele, vou fazer apenas algumas melhorias no design:

Edit in JSFiddle

- Result
- HTML
- CSS

Pronto, agora já podemos realizar a entrega do projeto. Caso tenha necessidade, o projeto pode ser baixado aqui.

Conheça também a **Formação Front End** da **Alura** e entenda um pouco mais a fundo desse mundo do Front End. :)

← Artigo Anterior

Ancorando elementos com HTML5

Próximo Artigo →

Testes em JavaScript

Leia também:

- Criando componentes CSS com o padrão BEM

- Organizando o CSS no seu projeto
- Nomes de classes no CSS
- Colocar as propriedades no CSS em ordem alfabética é melhor pra performance?
- CSS: animações com Transition e Animation
- Definindo a dimensão ideal para o layout do meu site
- CSS: Grids e tabelas com responsividade na Web

Veja outros artigos sobre
Front-end

Institucional

Sobre nós

Trabalhe conosco

Para Empresas

Para Escolas

Política de Privacidade

Política de Integridade

Termos de Uso

Status

A Alura

Como Funciona

Todos os cursos

Depoimentos

Instrutores(as)

Dev em <T>

Conteúdos

Alura Cases

Imersões

Artigos

Podcasts

Artigos de educação corporativa

Fale Conosco

Email e telefone

Perguntas frequentes

Novidades e Lançamentos

Seu e-mail

RECEBER

CURSOS

Cursos de Programação
Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

Cursos de Front-end
HTML, CSS | React | Angular | JavaScript | jQuery

Cursos de Data Science
Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

Cursos de DevOps
AWS | Azure | Docker | Segurança | IaC | Linux

Cursos de UX & Design
Usabilidade e UX | Vídeo e Motion | 3D

Cursos de Mobile
React Native | Flutter | iOS e Swift | Android, Kotlin | Jogos

Cursos de Inovação & Gestão
Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas