**GRID LAYOUT**

- Faz uma divisão com linhas e colunas (duas dimensões)

- É possível colocar elementos onde quiser nessa divisão

**### CONTAINER**

- display: grid

*Inicia o container, inicia o grid*

- grid-template-columns

*Defini quantas colunas o grid deve possuir*

- grid-template-rows

*Define quantas linhas o grid deve possuir*

- grid-template-gap

- grid-row-gap

Define o *espaçamento entre as linhas*

- grid-column-gap

Define o *espaçamento entre as colunas*

- grid-template-areas

É uma espécie de *matriz que define o posicionamento dos itens do grid*

**### ITEM(s)**

- grid-column

*Determina onde o item irá iniciar e terminar na coluna*

- grid-column-start

- grid-column-end

- grid-row

*Determina onde o item irá iniciar e terminar na linha*

- grid-row-start

- grid-row-end

- grid-area

*Determina qual será o posicionamento do item na matriz do grid*

**### REPRESENTAÇÃO SIMPLES NO HTML**

<!-- Elemento grid -->

<div class=”container”>

<!-- Itens do grid -->

<header></header>

<main></main >

<aside></aside>

<footer ></footer >

</div>

**### COLUNAS** (aplicação no grid)

.container {

display: grid;

grid-template-columns: 1fr 1fr;

ou

grid-template-columns: repeat(2, 1fr);

}

/\*

1fr 1fr divide em duas colunas iguais

1fr = 1 unidade fracional

- grid-template-columns: 3fr 1fr;

Faria a primeira coluna ser 3x maior que a segunda

\*/

**### LINHAS** (aplicação no grid)

.container {

display: grid;

grid-template-rows: 20vh 70vh 10vh;

}

/\*

Três linhas com alturas diferentes

vh = view height (altura da tela)

20vh = 20% da altura da tela

\*/

**### ESPAÇAMENTOS** (aplicação no grid)

.container {

grid-row-gap: 20px;

grid-column-gap: 50px;

ou

grid-gap: 20px 50px;

}

/\* Espaçamento entre as linhas e colunas \*/

**### POSICIONAMENTO** (aplicação no item)

.container header {

grid-column-start: 1;

grid-column-end: 3;

ou

grid-column: 1/3

}

/\*

Tendo o grid (div.container) três colunas o header, que é um elemento dentro do grid, irá compor todas as três, iniciando na primeira e terminando na terceira

\*/

.container footer {

grid-row-start: 3;

grid-row-end: 4;

ou

grid-row: 3/4

}

/\* Mesmo esquema das colunas \*/

**### POSICIONAMENTO COM TEMPLATE AREAS**

PRIMEIRAMENTE APLICANDO NO GRID

.container {

display: grid;

grid-template-areas: “header header”

“main aside”

“footer footer”;

/\*

Funciona como uma matriz para o posicionamento:

Neste caso seriam 2 colunas e 3 linhas.

Header ocupando a primeira linha e as duas colunas;

Main ocupando a segunda linha na primeira coluna;

Aside ocupando a segunda linha e a segunda coluna;

Footer ocupando a terceira linha e as duas colunas

\*/

ou

grid-template-areas: “h h”

“m a”

“f f”;

/\* Pode ser colocado qualquer nome \*/

ou ainda

grid-template-areas: “h .”

“m a”

“. f”;

/\*

O . (ponto) caracteriza um lugar vazio

Neste exemplo, na posição da primeira linha com a segunda coluna teria um espaço vazio, assim como na terceira linha com a primeira coluna

\*/

}

AGORA APLICANDO NOS ITENS DO GRID

.container header {

grid-area: header;

ou

grid-areas: h;

}

.container main {

grid-area: footer;

ou

grid-areas: f;

/\*

Pode ser colocado em qualquer posição

O nome não justifica a posição, até porque pode ser qualquer nome

\*/

**### ALINHAMENTO**

PROPRIEDADES

justify-content

align-content

justify-items

align-items

justify-self

align-self

- O grid é bidimensional, portanto, temos os eixos x e y

- Eixo x: posicionamento horizontal, da esquerda para a direita (justify)

- Eixo y: posicionamento vertical, de cima para baixo (align)

CONTENT (aplicação no grid)

justify-content e align-content nos permite alinhar o grid relativo ao espaço fora do próprio grid

O uso dessas propriedades é raro, pois só é aplicado caso o grid seja menor que a área definida, por exemplo: quando usamos px para o tamanho do grid e não fr (que é flexível), o grid pode ficar pequeno em relação ao tamanho real disponível do próprio grid (sua área)

Valores para **justify-content** (horizontal) e **align-content** (vertical)

start *posição no início*

end *posição no final*

center *posição ao centro*

stretch *ocupação do espaço disponível (padrão)*

space-between *espaçamento entro os itens*

space-around *espaçamento ao redor*

space-evenly *espaçamento contínuo (igual)*

<div class=”container”>

<div class=”item”></div>

<div class=”item”></div>

<div class=”item”></div>

</div>

.container {

justify-content: center;

align-content: center;

}

/\*

Alinha o grid ao centro

Para isso é necessário que o grid seja menor que o seu tamanho disponível (não esteja com a propriedade stretch)

\*/

ITEMS (aplicação no grid)

justify-items e align-items nos permite alinhar os itens do grid dentro do espaço disponível da célula que eles habitam

Para isso é necessário que o item tenha um tamanho menor que a própria célula, ou seja, que não preencha todo o espaço disponível dentro da célula (não esteja com a propriedade strectch)

Valores para **justify-items** (horizontal) e **align-items** (vertical)

start *posição no início*

end *posição no final*

center *posição ao centro*

stretch *ocupação do espaço disponível*

.container {

justify-items: center;

align-items: center;

ou

place-items: center;

}

/\*

Alinha os itens ao centro da sua célula, ou seja, ao centro do seu próprio espaço disponível dentro do grid

place-items centraliza nas duas direções ao mesmo tempo

\*/

SELF (aplicação no item)

justify-self e align-self nos permite alinhar o item em si

Faz a mesma coisa que o justify-items e o align-items, porém é aplicado diretamente no item do grid

Valores para **justify-self** (horizontal) e **align-self** (vertical)

start *posição no início*

end *posição no final*

center *posição ao centro*

stretch *ocupação do espaço disponível*

.container .item {

justify-self: center;

align-self: center;

}

/\*

Também alinha os itens ao centro da sua célula, ou seja, ao centro do seu próprio espaço disponível dentro do grid

\*/