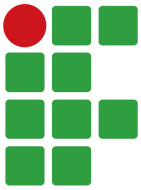

INSTITUTO FEDERAL
Espírito Santo
Campus de Alegre

**Apostila — Material de apoio**

# CSS Grid, Flexbox e calc()

*Da Teoria Fundamental à Prática de Mercado*

Desenvolvimento Front-End I

**Período:** TADS EaD - Semana 7

**Autor:** Prof. Cleziel Franzoni da Costa

Instituto Federal do Espírito Santo — Campus de Alegre

2025

# Sumário

# Capítulo 1

# [Complementar] - CSS Grid, Flexbox e calc()

## 1.1 Introdução: A Revolução do Layout em CSS

Por muitos anos, criar layouts complexos na web era uma tarefa árdua, dependente de *hacks* com `float`, `position` e tabelas. O CSS moderno, validado pelo W3C, introduziu dois sistemas que revolucionaram essa realidade: o **CSS Grid Layout** e o **CSS Flexible Box Layout (Flexbox)**.

Este guia é um manual de referência completo e aprofundado sobre essas duas tecnologias, além da poderosa função `calc()`. O objetivo é fornecer não apenas a sintaxe, mas também os conceitos, as estratégias e as boas práticas de mercado para que você possa construir qualquer layout de forma eficiente, responsiva e semântica.

## 1.2 CSS Grid Layout

O **CSS Grid Layout** é um sistema de layout **bidimensional** (controla colunas e linhas simultaneamente), projetado para dividir uma página ou um componente em regiões principais. É a ferramenta mais poderosa para o macro-layout de uma aplicação.

### 1.2.1 Conceitos Fundamentais do Grid

**Grid Container** O elemento pai onde aplicamos `display: grid`.

**Grid Item** Os filhos diretos do Grid Container.

**Grid Line** As linhas horizontais e verticais que dividem o grid, numeradas a partir de 1.

**Grid Track** O espaço entre duas linhas de grid, ou seja, uma coluna ou uma linha.

**Grid Cell** A menor unidade do grid, formada pela intersecção de uma linha e uma coluna.

**Grid Area** Um espaço retangular que pode ocupar uma ou mais células.

### 1.2.2 Propriedades para o Grid Container (Pai)

#### 1.2.2.1 Definindo a Estrutura do Grid

**`display`** Inicia o contexto do grid. Valores: `grid` ou `inline-grid`.

**`grid-template-columns`** Define o número e o tamanho das colunas.

**`grid-template-rows`** Define o número e o tamanho das linhas.

```
.container {
  display: grid;
  /* Cria 3 colunas: uma com 1fr, outra com 500px e a terceira com 1fr. */
  grid-template-columns: 1fr 500px 1fr;
  /* Cria 2 linhas: a primeira com altura fixa e a segunda com altura automtica.
      */
  grid-template-rows: 100px auto;
}
```

Listing 1.1: Exemplo básico de definição de um grid 3x2.

#### 1.2.2.2 Layout com Áreas Nomeadas

`grid-template-areas` permite criar um "mapa"visual do layout.

```
.container-areas {
  display: grid;
  grid-template-columns: 1fr 250px;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "header header"
    "main sidebar"
    "footer footer";
}

header { grid-area: header; }
main { grid-area: main; }
aside { grid-area: sidebar; }
footer { grid-area: footer; }
```

Listing 1.2: Definindo um layout semântico com grid-template-areas.

#### 1.2.2.3 Espaçamento (Gaps)

`gap` (shorthand para `row-gap` e `column-gap`) define o espaço entre as trilhas.

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px; /* 20px para linhas e colunas */
}
```

Listing 1.3: Aplicando um espaçamento de 20px entre todas as células.

#### 1.2.2.4 Alinhamento de Conteúdo

**`justify-items`** Alinha os itens **horizontalmente** dentro de suas células (`start`, `end`, `center`, `stretch`).

**`align-items`** Alinha os itens **verticalmente** dentro de suas células (`start`, `end`, `center`, `stretch`).

**`justify-content`** Alinha o **grid inteiro** horizontalmente no contêiner.

**`align-content`** Alinha o **grid inteiro** verticalmente no contêiner.

#### 1.2.2.5 Grid Implícito e Fluxo Automático

Quando há mais itens do que células definidas, o grid cria trilhas implícitas.

**`grid-auto-rows`** Define a altura das linhas criadas automaticamente.

**`grid-auto-columns`** Define a largura das colunas criadas automaticamente.

**`grid-auto-flow`** Controla como os itens automáticos são inseridos. O valor `dense` é poderoso para preencher buracos, mas pode alterar a ordem visual dos elementos.

### 1.2.3 Propriedades para os Grid Items (Filhos)

#### 1.2.3.1 Posicionamento Explícito

**`grid-column`** Shorthand para `grid-column-start` e `grid-column-end`.

**`grid-row`** Shorthand para `grid-row-start` e `grid-row-end`.

```
.item-destaque {
  /* Comea na linha de coluna 1 e ocupa 3 trilhas */
  grid-column: 1 / span 3;
```

```
  /* Comea na linha de linha 2 e termina na linha 4 */
  grid-row: 2 / 4;
}
```

Listing 1.4: Fazendo um item ocupar múltiplas células.

#### 1.2.3.2 Autoalinhamento Individual

**`justify-self`** Sobrescreve `justify-items` para um item específico.

**`align-self`** Sobrescreve `align-items` para um item específico.

### 1.2.4 Subgrid: O Próximo Nível de Alinhamento

O valor `subgrid` para `grid-template-columns` ou `grid-template-rows` permite que um item do grid, que também é um contêiner grid, herde a definição de trilhas de seu pai. Isso é revolucionário para alinhar itens aninhados.

```
.grid-pai {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  gap: 20px;
}
.card {
  grid-column: 1 / 4; /* Ocupa todas as colunas do pai */
  display: grid;
  /* O card herda as colunas e o gap do .grid-pai */
  grid-template-columns: subgrid;
  gap: inherit;
}
.card-title {
  grid-column: 2; /* Alinhado perfeitamente com a segunda coluna do .grid-pai
      */
}
```

Listing 1.5: Card com conteúdo alinhado ao grid pai usando subgrid.

**Nota:** O suporte para `subgrid` é recente. Verifique a compatibilidade em sites como "Can I use...".

## 1.3 CSS Flexbox

O **Flexible Box Layout (Flexbox)** é um sistema de layout **unidimensional** projetado para dar um controle preciso sobre o alinhamento e a distribuição de espaço entre itens em um contêiner.

### 1.3.1 Conceitos Fundamentais: Os Eixos

O Flexbox é governado por dois eixos: o Eixo Principal (*Main Axis*) e o Eixo Transversal (*Cross Axis*). A direção deles é definida pela propriedade `flex-direction`.

### 1.3.2 Propriedades para o Flex Container (Pai)

**`display`** Inicia o contexto Flexbox (`flex` ou `inline-flex`).

**`flex-direction`** Define a direção do eixo principal (`row`, `row-reverse`, `column`, `column-reverse`).

**`flex-wrap`** Define se os itens quebram de linha (`nowrap`, `wrap`, `wrap-reverse`).

**`justify-content`** Alinha os itens ao longo do **Eixo Principal**.

**`align-items`** Alinha os itens ao longo do **Eixo Transversal**.

**`align-content`** Alinha as **múltiplas linhas** de itens no Eixo Transversal (requer `flex-wrap: wrap`).

```
.nav-menu {
  display: flex;
  justify-content: space-between; /* Itens espaados horizontalmente */
  align-items: center; /* Itens centralizados verticalmente */
  flex-wrap: wrap; /* Em telas pequenas, os itens podem quebrar de linha */
  gap: 1rem;
}
```

Listing 1.6: Exemplo: um menu de navegação responsivo.

### 1.3.3 Propriedades para os Flex Items (Filhos)

**`order`** Altera a ordem visual de um item.

**`flex-grow`** A capacidade de um item crescer para ocupar espaço livre.

**`flex-shrink`** A capacidade de um item encolher se não houver espaço.

**`flex-basis`** O tamanho inicial de um item.

**`flex`** Shorthand para `flex-grow`, `flex-shrink` e `flex-basis`.

**`align-self`** Sobrescreve o `align-items` para um item específico.

```
.card {
  display: flex;
  flex-direction: column;
```

```
  height: 400px;
}
.card-content {
  flex-grow: 1; /* Faz este elemento ocupar todo o espao vertical disponvel */
}
.card-footer {
  flex-shrink: 0; /* Impede que o rodap encolha */
}
```

Listing 1.7: Um card com rodapé fixo na base, uma aplicação clássica de Flexbox.

## 1.4 A Função `calc()`

A função `calc()` permite realizar cálculos matemáticos para determinar valores de propriedades CSS, com a grande vantagem de poder **misturar unidades diferentes**.

### 1.4.1 Sintaxe e Regras de Uso

A sintaxe é `calc(expressão)`.

- Operadores permitidos: `+`, `-`, `*`, `/`.
- **Regra Crítica:** Os espaços ao redor dos operadores de adição (`+`) e subtração (`-`) são **obrigatórios**.

### 1.4.2 Casos de Uso Práticos e Exemplos

```
/* 1. Largura total com padding fixo */
.container {
  width: calc(100% - 40px);
}

/* 2. Altura total da tela menos um cabealho fixo */
.main-content {
  min-height: calc(100vh - var(--header-height));
}

/* 3. Tipografia fluida com tamanho mnimo */
h1 {
  font-size: calc(18px + 1.5vw);
}

/* 4. Criar um grid com uma coluna de largura calculada */
.grid-container {
```

```
  display: grid;
  grid-template-columns: 250px calc(100% - 270px);
  gap: 20px;
}
```

Listing 1.8: Exemplos variados do uso de calc().

## 1.5 Estratégias Avançadas e Boas Práticas de Mercado

### 1.5.1 Grid vs. Flexbox: A Escolha Estratégica

A regra de ouro do mercado não é "ou um ou outro", mas "os dois juntos".

| Critério | Flexbox | Grid |
|---|---|---|
| **Dimensão** | **Unidimensional**: Otimizado para alinhar itens em uma única linha OU uma única coluna. | **Bidimensional**: Otimizado para alinhar itens em linhas E colunas simultaneamente. |
| **Foco** | **Conteúdo**: O layout é definido pelo tamanho e fluxo do conteúdo. É "de dentro para fora". | **Layout**: O layout é definido no contêiner, e os itens são posicionados nele. É "de fora para dentro". |
| **Casos de Uso** | Menus de navegação, listas de cards, formulários, alinhamento de ícones, componentes de UI. | Layout geral da página, dashboards, galerias de imagens complexas, estruturas de componentes com alinhamento em duas direções. |

### 1.5.2 Padrões de Layout Essenciais

#### 1.5.2.1 Centralização Perfeita

```
/* Com Flexbox */
.container-flex {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh; /* Para centralizar na pgina inteira */
}
```

```css
/* Com Grid (ainda mais conciso) */
.container-grid {
  display: grid;
  place-items: center; /* Shorthand para justify-items e align-items */
  min-height: 100vh;
}
```

Listing 1.9: A forma moderna e definitiva de centralizar um item.

#### 1.5.2.2 Responsividade Inteligente com Grid

Este é o padrão mais celebrado para galerias responsivas sem media queries.

```css
.gallery {
  display: grid;
  gap: 1rem;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
}
```

Listing 1.10: O padrão "auto-fit, minmax"para responsividade automática.

**Como funciona:** O grid cria quantas colunas de `300px` no mínimo couberem. Se sobrar espaço, elas crescem. Se faltar, os itens quebram para a próxima linha automaticamente.

#### 1.5.2.3 O Truque do 'margin: auto' em Flexbox

Uma forma poderosa de empurrar itens para as extremidades.

```css
.nav-menu {
  display: flex;
}
.nav-item:last-child {
  margin-left: auto; /* Mgica! */
}
```

Listing 1.11: Empurrando o último item de um menu para a direita.

### 1.5.3 Acessibilidade (A11y) e a Ordem do DOM

Tanto Grid quanto Flexbox (`order`) podem alterar a ordem visual dos elementos, separando-a da ordem do código HTML (DOM).

- **Alerta:** Leitores de tela e a navegação por teclado (tecla `Tab`) seguem a ordem do DOM.
- **Boa Prática:** Sempre que possível, mantenha uma ordem lógica no HTML que corresponda à ordem visual. Use o poder de reordenação do CSS com extrema

cautela e apenas para aprimoramentos visuais que não quebrem a experiência de navegação sequencial.

### 1.5.4 Dicas de Depuração (Debugging)

Os navegadores modernos possuem ferramentas fantásticas para depurar Grid e Flexbox.

- Abra o **DevTools** (F12 ou Inspecionar).
- No painel de Elementos, encontre seu contêiner. Você verá um *badge* escrito "grid"ou "flex".
- Clique no *badge* para ativar um *overlay* visual na página, que mostra as linhas, trilhas, gaps e eixos.
- O painel de Layout no DevTools oferece opções detalhadas para customizar a visualização. Use-o extensivamente!