Ebook — CSS Base, Flex e Grid

Objetivo: estilizar o HTML do exercício em três camadas: **Base (base.css)** → **Layout Flex (flex.css)** → **Componentes (base.css)** → **Layout Grid (grid.css)**, entendendo o papel de cada arquivo e como alternar entre Flex e Grid sem mudar o HTML.

1) Preparação do projeto

Estrutura de pastas/arquivos

```
/projeto
  ├─ index.html
  ├─ base.css
  ├─ flex.css
  ├─ grid.css
  └─ script.js
```

Links no `<head>` do HTML

- Base sempre incluída.
- Escolha **um** layout por vez: `flex.css` **ou** `grid.css`.

Exemplo (usando Grid):

```
<link rel="stylesheet" href="base.css">
<!-- <link rel="stylesheet" href="flex.css"> -->
<link rel="stylesheet" href="grid.css">
```

Para testar o **Flex**, troque:

```
<link rel="stylesheet" href="base.css">
<link rel="stylesheet" href="flex.css">
<!-- <link rel="stylesheet" href="grid.css"> -->
```

2) Base CSS (fundação visual + tokens)

O **base.css** define tokens (variáveis), reset/normalização simples, tipografia, cores, espaçamentos e estilos **genéricos**.

Ele **não** escolhe se o layout principal é Flex ou Grid — isso fica em `flex.css` ou `grid.css`.

2.1 — Código (base.css)

```css
:root{
  --main-radius: 5px;
  --main-padding: 5px;
  --gap: 5px;
  --bg: #8c7458;
  --cor-menu: #333;
  --cor-menu-hover: #fff;
}

/* Reset essencial + box model previsível */
*,
*::before,
*::after { box-sizing: border-box; }
```

```css
body{
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
  color: #fff;
  background-color: #f2f2f2;
  text-align: center;
}

/* Largura máxima e centralização do site */
.container{
  max-width: 1200px;
  margin: 0 auto;
}
```

```css
header, aside, main, article, footer{
  background-color: var(--bg);
  margin: var(--gap);
  border-radius: var(--main-radius);
  padding: var(--main-padding);
}
```

```css
li{ list-style: none; margin: 0; padding: 0; }
a{ color: inherit; }
```

2.2 — Por que assim?

- **Variáveis (`:root`)** centralizam decisões de cor, raio e espaçamento (fácil de manter).
- `box-sizing: border-box;` evita surpresas de cálculo de largura/altura.
- `max-width + margin: 0 auto` limita a largura e centraliza o conteúdo.
- **Estilos por tags semânticas** dão aparência consistente a grandes blocos (mas serão ajustados nos layouts).

3) Layout com Flex (flex.css)

O **flex.css** organiza **a página** em colunas/linhas com Flexbox. É ótimo para layouts lineares e rápidas distribuições.

3.1 — Código (flex.css)

```css
.container{
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}
```

```css
.main-container{
  display: flex;
  flex: 1;
}
```

```css
aside{
  flex-basis: 20%;
  flex-shrink: 0;
}
```

```css
.content-container{
  display: flex;
  flex-direction: column;
  flex: 1;
}
```

```css
main{
  flex: 1;
  padding: 0;
}
```

```css
.content-row{
  display: flex;
  gap: 10px;
}
```

```css
.content{
  flex: 1 1 0;
}
```

3.2 — O que observar

- `.container` em coluna cria **Topo (header)** → **Miolo (main-container)** → **Base (footer)** ocupando toda a altura da tela.
- `.main-container` lado a lado: **aside** (20%) + **content-container** (80%).
- `.content-row` usa Flex para **três cards** fluidos.
- Definimos `padding: 0` em `main` porque o **base.css** já dá o “acolchoamento” nos blocos; aqui ajustamos para o layout.

4) Volta ao Base: componentes (menu, busca, carrossel, cards, rodapé)

Componentes são “ilhas” que podem usar **Flex internamente** sem interferir na decisão Flex/Grid do **layout principal**.

4.1 — Menu e navegação (em base.css)

```css
.menu{
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 0 10px;
}

.logo{
  font-weight: bold;
  font-size: 20px;
}

.menu-items{
  display: flex;
  justify-content: center;
  gap: 15px;
  padding: 0;
  margin: 0;
}

.menu-items a,
.author-email,
footer ul a{
  text-decoration: none;
  color: var(--cor-menu);
  font-weight: bold;
}

.menu-items a:hover,
footer ul a:hover,
.author-email:hover{
  text-decoration: underline;
  color: var(--cor-menu-hover);
}
```

4.2 — Busca (em base.css)

```css
.search-bar{
  display: flex;
  align-items: center;
  padding: 8px;
}
input{
  margin-right: 5px;
  padding: 6px 8px;
  border-radius: 5px;
  border: 1px solid #cbb79f;
  background-color: #e9d8c5;
}
button{
  cursor: pointer;
  padding: 6px 12px;
  border-radius: 5px;
  border: 1px solid transparent;
  background-color: #fff;
  color: #333;
  font-weight: bold;
}
button:hover{
  background-color: #333;
  color: #fff;
}
```

4.3 — Carrossel (em base.css)

```css
.carousel {
  position: relative;
  max-width: 950px;
  margin: 0 auto;
  overflow: hidden;
  border-radius: var(--main-radius);
}
.carousel-images {
  display: flex;
  transition: transform 0.5s ease-in-out;
}
```

```css
.carousel-images img {
  width: 100%;
  height: auto;
  flex-shrink: 0;
}
```

```css
.prev, .next{
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  border: none;
  padding: 0;
  width: 44px;
  height: 44px;
  display: grid;
  place-items: center;
  border-radius: 50%;
  background-color: rgba(0,0,0,0.5);
  color: #fff;
  cursor: pointer;
  z-index: 2;                /* <- acima da imagem */
}
.prev { left: 10px; }
.next { right: 10px; }
```

4.4 — Conteúdo e cards (em base.css)

```css
.content{ padding: 20px; }
.content img{ border-radius: 10px; width: 100%; height: auto; }
.content p{
  font-size: 12px;
  text-align: justify;
  margin: 20px;
}
```

4.5 — Rodapé (em base.css)

```css
.footer-container{
  display: flex;
  justify-content: space-evenly;
  flex-wrap: wrap;
  gap: 10px;
}
```

```css
.footer-container ul,
.footer-container div{
  flex: 1;
  min-width: 180px;
  max-width: 30%;
}
footer ul{ padding: 0; margin: 0; }
footer ul li{ padding-bottom: 5px; }

.footer-bottom p{
  margin: 8px 0;
}
```

5) Layout com Grid (grid.css)

O **grid.css** resolve o layout principal com **CSS Grid**, perfeito para colunas/linhas explícitas e alinhamento robusto.

5.1 — Código (grid.css)

```css
.container{
  display: grid;
  grid-template-rows: auto 1fr auto; /* header, main-container, footer */
  min-height: 100vh;
}
```

```css
.main-container{
  display: grid;
  grid-template-columns: 1fr 3fr;   /* ~20% | 80% */
  gap: var(--gap);
  align-items: stretch;
}
```

```css
.main-container > aside,
.main-container > .content-container{
  margin: 0;
}

/* Conteúdo interno em grid: main em cima, cards embaixo */
```

```css
.content-container{
  display: grid;
  grid-template-rows: auto 1fr;
  gap: var(--gap);
}
/* Zera margens dos filhos diretos aqui também */
.content-container > main,
.content-container > .content-row{
  margin: 0;
  padding: 0;
}

/* Cards em grade */
.content-row{
  display: grid;
  grid-template-columns: repeat(3, minmax(0, 1fr));
  gap: 10px;
}
```

5.2 — Por que “zerar margens”?

- O **base.css** coloca `margin: var(--gap)` em `aside, main, article`, etc.
- Em Grid, usamos `gap` para espaçamentos. Se mantivermos **margens + gap**, aparecem “quebras” e alturas desalinhadas.
- Por isso, removemos `margin` **somente** dos **filhos diretos** do grid (`.main-container` e `.content-container`) para deixar o `gap` trabalhar.

5.3 — Dica para o “aside que não vai até o fim”

- `align-items: stretch;` na `.main-container` + remoção de margens dos filhos diretos garantem que **aside** e **content-container** estiquem uniformemente.

6) Como alternar entre Flex e Grid (sem tocar no HTML)

1. No `<head>`, comente/descomente os links:

Flex:

```
 <link rel="stylesheet" href="base.css">
<link rel="stylesheet" href="flex.css">
<!-- <link rel="stylesheet" href="grid.css"> -->
```

Grid:

```
 <link rel="stylesheet" href="base.css">
<!-- <link rel="stylesheet" href="flex.css"> -->
<link rel="stylesheet" href="grid.css">
```

2. Recarregue a página e observe:

- Distribuição de **cards** (linha flexível vs. grade explícita).
- **Alturas** do `aside` e do `content-container`
- Espaçamentos controlados por **gap** no Grid.

7) Checklist rápido

- **base.css** carregado antes do arquivo de layout.
- **Somente um** layout ativo (Flex **ou** Grid).
- Componentes (menu, busca, carrossel, rodapé) funcionam com **qualquer** layout.
- Em Grid, sem “saltos” de margem (margens zeradas nos **filhos diretos** das áreas de Grid).
- Imagens com `width: 100%; height: auto;` para manter proporção.

8) Erros comuns e soluções

- **“O aside não estica no Grid”**
  → Confirme `align-items: stretch;` na `.main-container` e remova `margin` dos filhos diretos.
- **“Espaço duplo entre colunas”**
  → Em Grid, prefira `gap`; **evite** `margin` nos filhos diretos das áreas de Grid.

- **“Menu desalinhado”**

  → Verifique `.menu { display:flex; align-items:center; justify-content:space-between; }` e `gap` nos itens.

- **“Cards quebram no Flex”**

  → `flex: 1 1 0;` nos cards permite distribuição uniforme; ajuste `gap` no container.