CSS GRID LAYOUT

Themen: - CSS Grid Grundlagen - Grid Container und Items - Grid Lines, Tracks und Areas - Responsive Grid Layouts - Grid vs. Flexbox - Praktische Layout-Patterns

Was ist CSS Grid?

CSS Grid Layout

CSS Grid ist ein **zweidimensionales** Layout-System für das Web - es kann sowohl Zeilen als auch Spalten gleichzeitig handhaben.

Hauptkonzepte:

- **Grid Container**: Das Elternelement mit `display: grid`
- **Grid Items**: Die direkten Kindelemente des Containers
- **Grid Lines**: Die Trennlinien zwischen Tracks
- **Grid Tracks**: Die Zeilen und Spalten
- **Grid Areas**: Benannte Bereiche im Grid

Was ist CSS Grid?

Grundlegendes Beispiel:

```css
.container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    grid-template-rows: 100px 200px;
    gap: 10px;
}
```

Resultat: 3 Spalten, 2 Zeilen mit Abstand

Grid Container Eigenschaften I

Grid Container definieren:

`display: grid | inline-grid` - Aktiviert Grid Layout für das Element

`grid-template-columns` & `grid-template-rows`

```
.grid {
    /* 3 gleiche Spalten */
    grid-template-columns: 1fr 1fr 1fr;
    /* oder kürzer: */
    grid-template-columns: repeat(3, 1fr);

    /* Verschiedene Größen */
    grid-template-columns: 200px 1fr 100px;

    /* Auto-Größe basierend auf Inhalt */
    grid-template-columns: auto 1fr auto;
}
```

Einheiten: - `fr` (fraction): Teilt verfügbaren Platz auf - `px`, `em`, `%`: Feste Größen - `auto`: Größe basierend auf Inhalt - `minmax()`: Flexible Bereiche

Grid Container Eigenschaften II

Abstände:

gap (früher grid-gap)

```
.grid {
    gap: 20px;                /* Alle Abstände */
    gap: 10px 20px;           /* row-gap column-gap */
    row-gap: 10px;            /* Nur Zeilen */
    column-gap: 20px;         /* Nur Spalten */
}
```

justify-items & align-items

```
.grid {
    justify-items: start | center | end | stretch;   /* horizontal */
    align-items: start | center | end | stretch;     /* vertikal */
    place-items: center;      /* Kurzform für beide */
}
```

Grid Container Eigenschaften II

Ausrichtung:

justify-items & align-items

```
.grid {
    justify-items: start | center | end | stretch;  /* horizontal */
    align-items: start | center | end | stretch;    /* vertikal */
    place-items: center;    /* Kurzform für beide */
}
```

justify-content & align-content

```
.grid {
    justify-content: start | center | end | space-between | space-around;
    align-content: start | center | end | space-between | space-around;
}
```

Grid Items positionieren

Mit Grid Lines (Linien-basiert):

```
.item {
    grid-column-start: 1;
    grid-column-end: 3;      /* Von Linie 1 bis 3 */
    /* oder kürzer: */
    grid-column: 1 / 3;      /* Spalten 1-2 */
    grid-row: 2 / 4;         /* Zeilen 2-3 */

    /* Span-Notation */
    grid-column: 1 / span 2; /* Start bei 1, erstreckt sich
über 2 Spalten */
    grid-row: span 3;        /* Erstreckt sich über 3 Zeilen */
}
```

Grid Items positionieren

Kurzform:

```
.item {
    grid-area: 2 / 1 / 4 / 3;  /* row-start / col-start / row-end / col-end */
}
```

Beispiel HTML:

```
<div class="grid">
    <div class="item header">Header</div>
    <div class="item content">Content</div>
    <div class="item sidebar">Sidebar</div>
</div>
```

Named Grid Areas

Benannte Grid-Bereiche:

Grid Areas machen Layouts lesbar und wartbar.

CSS:

```css
.grid {
    display: grid;
    grid-template-columns: 1fr 3fr 1fr;
    grid-template-rows: auto 1fr auto;
    grid-template-areas:
        "header  header  header"
        "sidebar main    ads"
        "footer  footer  footer";
    gap: 20px;
}

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

HTML:

```html
<div class="grid">
    <header class="header">Header</header>
    <aside class="sidebar">Sidebar</aside>
    <main class="main">Main Content</main>
    <aside class="ads">Advertisements</aside>
    <footer class="footer">Footer</footer>
</div>
```

Auto-Grid und minmax()

Automatische Grid-Erstellung:

`grid-auto-rows` & `grid-auto-columns`

```
.grid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-auto-rows: 200px;  /* Neue Zeilen haben 200px Höhe */
}
```

`minmax()` für flexible Größen:

```
.grid {
    grid-template-columns: minmax(200px, 1fr) minmax(300px,
2fr);
    grid-auto-rows: minmax(100px, auto);
}
```

Auto-Grid und minmax()

Auto-fit und Auto-fill:

```
/* Auto-fit: Spalten kollabieren wenn leer */
.grid {
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
}

/* Auto-fill: Spalten bleiben auch wenn leer */
.grid {
    grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
}
```

Perfekt für responsive Card-Layouts ohne Media Queries!

Grid vs. Flexbox

Aspekt	**Flexbox**	**CSS Grid**
Dimensionen	1D (eine Richtung)	2D (Zeilen + Spalten)
Verwendung	Komponenten-Layout	Seiten-Layout
Content	Content-First	Layout-First
Ausrichtung	Entlang einer Achse	In beide Richtungen

Flexbox verwenden für: - Navigation bars - Button groups - Zentrieren von Elementen - Gleichmäßige Verteilung

Grid verwenden für: - Seitenlayout (Header, Main, Footer) - Card grids - Dashboard layouts - Komplexe responsive Designs

Sie können kombiniert werden!

```
.page-grid { display: grid; }        /* Seitenlayout */
.nav-flex { display: flex; }         /* Navigation innerhalb des Grids */
```

Responsive Grid Patterns I

Pattern 1: Responsive Card Grid

```
.card-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px,
1fr));
    gap: 2rem;
}
```

Resultat: Cards brechen automatisch um wenn Platz nicht reicht!

Responsive Grid Patterns I

Pattern 2: Holy Grail Layout

```
.holy-grail {
    display: grid;
    min-height: 100vh;
    grid-template-columns: 200px 1fr 200px;
    grid-template-rows: auto 1fr auto;
    grid-template-areas:
        "header header header"
        "sidebar main ads"
        "footer footer footer";
}

@media (max-width: 768px) {
    .holy-grail {
        grid-template-columns: 1fr;
        grid-template-areas:
            "header"
            "main"
            "sidebar"
            "ads"
            "footer";
    }
}
```

Responsive Grid Patterns II

Pattern 3: Dashboard Layout

```css
.dashboard {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-auto-rows: minmax(200px, auto);
    gap: 1rem;
}

.widget-large {
    grid-column: span 2;
    grid-row: span 2;
}
.widget-wide {
    grid-column: span 2;
}
.widget-tall {
    grid-row: span 2;
}
```

```css
@media (max-width: 1024px) {
    .dashboard {
        grid-template-columns: repeat(2, 1fr);
    }
    .widget-large { grid-column: span 1; }
}
```

Responsive Grid Patterns II

Pattern 4: Magazine Layout

```
.magazine {
    display: grid;
    grid-template-columns: repeat(6, 1fr);
    grid-auto-rows: 200px;
    gap: 1rem;
}

.article-featured { grid-column: span 4; grid-row: span 2; }
.article-large { grid-column: span 3; }
.article-small { grid-column: span 2; }
```

Advanced Grid Features

Subgrid (CSS Grid Level 2):

```
.main-grid {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
}

.nested-grid {
    display: grid;
    grid-template-columns: subgrid;  /* Erbt Spalten vom Parent
*/
    grid-column: span 3;
}
```

Advanced Grid Features

Grid Functions:

```
.grid {
    /* fit-content: Zwischen min-content und max-content */
    grid-template-columns: fit-content(300px) 1fr;

    /* repeat mit verschiedenen Patterns */
    grid-template-columns: repeat(auto-fit, minmax(200px,
1fr));

    /* min/max für responsive Designs */
    grid-template-rows: repeat(3, minmax(100px, max-content));
}
```

Advanced Grid Features

Dense Placement:

```
.masonry-style {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px,
1fr));
    grid-auto-flow: row dense;  /* Füllt Lücken automatisch */
}
```

Praktische Tipps & Best Practices

✅ **DO:** - Verwende **Grid für Layout**, **Flexbox für Komponenten** - Nutze `fr` Units für flexible Spalten - Verwende `gap` statt margins für Abstände - Benenne Grid Areas für komplexe Layouts - Nutze `auto-fit/auto-fill` für responsive Designs

❌ **DON'T:** - Überkomplizierte Nested Grids vermeiden - Nicht jeden Container zu einem Grid machen - `position: absolute` innerhalb von Grids vermeiden - Alte Browser ohne Fallbacks ignorieren

Debugging-Tipps: - **Firefox Grid Inspector** zeigt Grid Lines - **Chrome DevTools** visualisiert Grid Areas - `grid-template-areas` macht Code selbsterklärend

Browser Support Check:

```
@supports (display: grid) {
    .grid-layout { display: grid; }
}
```

Performance-Tipp: Grid ist sehr performant - keine Sorge vor komplexen Layouts!

Vollständiges Beispiel: Responsive Blog Layout

```html
<div class="blog-layout">
    <header class="header">My Blog</header>
    <nav class="nav">Navigation</nav>
    <main class="content">
        <article>Article 1</article>
        <article>Article 2</article>
        <article>Article 3</article>
    </main>
    <aside class="sidebar">Sidebar</aside>
    <footer class="footer">Footer</footer>
</div>
```

```css
.blog-layout {
    display: grid;
    min-height: 100vh;
    grid-template-columns: 200px 1fr 300px;
    grid-template-rows: auto auto 1fr auto;
    grid-template-areas:
        "header header header"
        "nav nav nav"
        "sidebar content sidebar-right"
        "footer footer footer";
    gap: 1rem;
}

.content {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 2rem;
}
```

Ressourcen: - CSS Grid Complete Guide - Grid by Example - MDN CSS Grid Layout