CSS Grid Layout

A Comprehensive Study Material with Theory, Patterns, and Hands-on Exercises

Aditya Tandon

September 29, 2025

Overview

This section is a deep dive into CSS Grid—a modern, two-dimensional layout system for the web. You will learn core concepts, sizing strategies, explicit vs. implicit grids, line- and area-based placement, alignment, responsive patterns, and advanced techniques such as overlapping and nested grids. We end with practical layout case studies and a set of graded exercises.

Prerequisites

Comfort with HTML structure, basic CSS selectors and properties, and using a browser's Developer Tools.

1 Why CSS Grid? A Short History of Layout

Web layout has evolved through several eras:

- **Tables (1990s):** misused for page layout; coupled content and presentation.
- **Floats (2000s):** allowed multi-column layouts but required clearfix hacks and were fragile.
- **Positioning:** powerful but verbose for full-page layout.
- **Flexbox (2010s):** one-dimensional (row *or* column) alignment system; excellent for components.
- **Grid (2017+):** *two-dimensional* layout (rows *and* columns) with explicit tracks and areas.

Grid is designed for page-level layout; Flexbox shines for arranging items along a single axis (e.g., nav bars, button groups). In practice, we *combine* Grid (macro layout) and

Flexbox (micro layout).

2 Mental Model and Terminology

A grid partitions space into rows and columns. Key terms (see MDN Web Docs):

- **Grid container:** element with `display: grid | inline-grid`.
- **Grid items:** the container's direct children.
- **Grid line:** the boundary lines defining tracks (numbered).
- **Grid track:** a row or column (space between two lines).
- **Grid cell:** the intersection of one row track and one column track.
- **Grid area:** a rectangular set of cells (one or more).

ASCII diagram (3 columns, 2 rows).

```
Lines:   1       2       3       4
Cols:    |  col1  |  col2  |  col3  |
Rows:
 1 ---- +--------+--------+--------+
        |  (1,1) |  (1,2) |  (1,3) |
 2 ---- +--------+--------+--------+
        |  (2,1) |  (2,2) |  (2,3) |
 3 ---- +--------+--------+--------+
```

3 Creating a Grid

3.1 Becoming a Grid Container

Listing 1: Grid container

```
.grid { display: grid; } /* block-level grid */
.inlineGrid { display: inline-grid; } /* inline-level grid */
```

3.2 Defining Columns and Rows

Listing 2: Fixed tracks

```
.grid {
  display: grid;
  grid-template-columns: 200px 1fr 200px; /* left fixed, flexible
     middle, right fixed */
  grid-template-rows: 120px auto 80px; /* header, content, footer */
```

```
  gap: 16px; /* row + column gap */
}
```

The `fr` unit divides *remaining* space fractionally. `auto` sizes track to content or leftover rules.

3.3 Repeat and Minmax

Listing 3: repeat() and minmax()

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr); /* 3 equal columns */
  grid-auto-rows: minmax(120px, auto); /* implicit rows have a minimum
      height */
}
```

`repeat()` reduces repetition. `minmax(min,max)` bounds track size.

3.4 Content-based Sizing

Listing 4: min-content, max-content, fit-content()

```
.columns {
  display: grid;
  grid-template-columns: min-content max-content fit-content(18ch) 1fr;
}
```

min-content shrinks to the longest unbreakable piece; **max-content** expands to fit content; **fit-content(X)** clamps to at most `X`.

4 Explicit vs. Implicit Grids

Explicit grid is defined by `grid-template-rows/columns`. Items beyond those tracks spill into the **implicit grid**.

Listing 5: Implicit rows via grid-auto-rows

```
.gallery {
  display: grid;
  grid-template-columns: repeat(4, 1fr); /* explicit 4 columns */
  grid-auto-rows: 180px; /* any extra rows get this height */
  gap: 12px;
}
```

5 Auto-placement and Flow

`grid-auto-flow` affects how unpositioned items are placed:

- **row (default):** fill rows left-to-right, then new rows.
- **column:** fill columns top-to-bottom, then new columns.
- **dense:** backfill earlier holes (may change visual order).

Listing 6: Auto-flow modes

```
.autoRow { grid-auto-flow: row; }
.autoCol { grid-auto-flow: column; }
.autoDense { grid-auto-flow: row dense; }
```

Note: Dense packing can compromise source order semantics; use with care.

6 Gaps and Alignment

6.1 Gap Shorthand

```
.grid { gap: 16px; } /* row-gap and column-gap are 16px */
.grid { row-gap: 8px; column-gap: 24px; }
```

6.2 Aligning the Whole Grid vs. Items

- **justify-content** / **align-content:** position the grid within its container (when extra space).
- **justify-items** / **align-items:** default alignment of items within their cells.
- **justify-self** / **align-self:** per-item override.

Listing 7: Box alignment in Grid

```
.grid {
  justify-content: center; /* center the grid as a whole */
  align-content: start; /* stick to top if extra vertical space */
  justify-items: stretch; /* items fill cell horizontally */
  align-items: center; /* items centered vertically */
}
.card:nth-child(3) { align-self: start; } /* override one item */
```

Shorthands: `place-content: <align-content> <justify-content>;` and `place-items: <align-items> <justify-items>;`.

7 Placing Items: Lines vs. Areas

7.1 Line-based Placement

Listing 8: Span by lines

```
.feature { grid-column: 1 / 3; } /* from line 1 to 3 => spans 2 columns */
.sidebar { grid-row: 1 / span 2; } /* spans two rows starting at line 1 */
```

7.2 Naming Lines for Readability

Listing 9: Named lines

```
.layout {
  display: grid;
  grid-template-columns: [side-start] 240px [side-end content-start] 1 fr [content-end];
  grid-template-rows: [head-start] 80px [head-end main-start] 1fr [main-end foot-start] 100px [foot-end];
}
.sidebar { grid-column: side-start / side-end; }
.main { grid-column: content-start / content-end; }
```

7.3 Area-based Placement

Listing 10: grid-template-areas

```
.page {
  display: grid;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
  grid-template-columns: 260px 1fr;
  grid-template-rows: 90px 1fr 80px;
  gap: 16px;
}
.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
.footer { grid-area: footer; }
```

Areas are intuitive for wireframes and quick rearrangements.

8 Overlapping, Layering, and Z-index

Grid allows overlap by assigning intersecting areas. Use `z-index` to control stacking order.

Listing 11: Overlap example

```
.hero {
  grid-area: 1 / 1 / 3 / 3; /* large background area */
}
.cta {
  grid-area: 2 / 2 / 3 / 3; /* on top inside bottom-right cell */
  z-index: 2;
}
```

9 Nested Grids and Subgrid

9.1 Nested Grids

Any grid item can be a grid container for its children.

Listing 12: Nested grid card

```
.card {
  display: grid;
  grid-template-rows: auto 1fr auto; /* image, body, footer */
  gap: 8px;
}
```

9.2 Subgrid (Support Note)

`subgrid` lets child grid tracks align with the parent's tracks. As of writing, support is improving (Firefox supports subgrid for rows/columns; check current compatibility).[1]

10 Responsive Grid Patterns

10.1 Auto-fit and Auto-fill

Listing 13: Fluid card grid

```
.cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
  gap: 16px;
```

[1] See MDN: MDN Web Docs.

```
}
```

auto-fit collapses empty tracks—items stretch to fill rows. **auto-fill** preserves track slots (useful when you want gutters to remain).

10.2 Media Queries with Grid

Listing 14: Responsive reflow

```
.catalog {
  display: grid; gap: 16px;
  grid-template-columns: 1fr 2fr; /* desktop: sidebar + content */
  grid-template-areas:
    "filters list";
}
@media (max-width: 800px) {
  .catalog {
    grid-template-columns: 1fr;
    grid-template-areas:
      "filters"
      "list"; /* stack on mobile */
  }
}
```

11 Debugging and Best Practices

11.1 DevTools Grid Inspector

Enable *Grid overlays* in Firefox/Chrome DevTools to visualize lines, track numbers, and areas. Toggle track numbers to debug placement.

11.2 Accessibility and Source Order

Prefer a DOM order that matches reading order. Avoid using grid purely to reorder content in a way that confuses screen readers.

11.3 When to Use Grid vs. Flexbox

- Use **Grid** for two-dimensional page layout or card grids.
- Use **Flexbox** for one-dimensional alignment (nav bars, toolbars, chips).
- Combine: Grid for macro structure, Flexbox inside components.

12 Case Studies (Step-by-step)

12.1 Case Study A: Responsive Image Gallery

HTML

```
<div class="gallery">
  <img src="img/pic1.jpg" alt="Pic 1">
  <img src="img/pic2.jpg" alt="Pic 2">
  <img src="img/pic3.jpg" alt="Pic 3">
  <img src="img/pic4.jpg" alt="Pic 4">
  <img src="img/pic5.jpg" alt="Pic 5">
</div>
```

CSS

```
.gallery {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(160px, 1fr));
  gap: 12px;
}
.gallery img {
  width: 100%;
  height: 160px;
  object-fit: cover; /* keep aspect while filling box */
  border-radius: 8px;
}
```

Why it works: `auto-fit` packs images per row based on available width; `minmax()` ensures usable thumbnails.

12.2 Case Study B: Product Cards Grid

HTML

```
<section class="products">
  <article class="card">
    <img src="img/a.jpg" alt="Product A">
    <h3>Product A</h3>
    <p class="price">&#8377; 1,499</p>
    <button>Add to Cart</button>
  </article>
  <!-- repeat cards -->
```

```
</section>
```

CSS

```
.products {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(220px, 1fr));
  gap: 18px;
}
.card {
  display: grid;
  grid-template-rows: auto auto 1fr auto;
  gap: 8px;
  padding: 14px;
  border: 1px solid #e5e7eb; border-radius: 10px;
}
.card img { width: 100%; height: 140px; object-fit: cover; border-
   radius: 8px; }
.price { color: #065f46; font-weight: 700; }
```

Notes: Using a nested grid in `.card` vertically arranges image, title, flexible description, and button.

12.3 Case Study C: Magazine / News Layout

CSS

```
.mag {
  display: grid;
  grid-template-areas:
    "lead lead side"
    "lead lead side"
    "grid grid grid";
  grid-template-columns: 2fr 2fr 1fr;
  grid-template-rows: repeat(2, 220px) auto;
  gap: 16px;
}
.lead { grid-area: lead; background: #111; color: #fff; padding: 16px;
    }
.side { grid-area: side; background: #f3f4f6; padding: 12px; }
.grid { grid-area: grid;
```

```
  display: grid; gap: 12px;
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
}
```

Idea: Feature story spans two rows; sidebar stays fixed; below is a responsive article grid.

12.4 Case Study D: The Holy Grail Layout

CSS

```
.holy {
  display: grid; gap: 12px;
  grid-template-areas:
    "head head"
    "side main"
    "foot foot";
  grid-template-columns: 260px 1fr;
  grid-template-rows: 90px 1fr 80px;
  min-height: 100vh;
}
.head { grid-area: head; background: #1f2937; color:#fff; }
.side { grid-area: side; background: #f1f5f9; }
.main { grid-area: main; }
.foot { grid-area: foot; background: #111827; color:#fff; }

@media (max-width: 800px) {
  .holy {
    grid-template-columns: 1fr;
    grid-template-areas:
      "head"
      "main"
      "side"
      "foot";
  }
}
```

13 Common Pitfalls (and Fixes)

- **"My grid isn't showing columns."** Did you set `grid-template-columns`? With only `display:grid`, the grid is one column.
- **Overflowing content.** Consider `minmax()` or `fit-content()` to bound track growth;

set `object-fit` for images.

- **Unexpected wrapping.** Check implicit grid settings: `grid-auto-rows/columns`.
- **Alignment confusion.** Distinguish between *items* vs. *content* alignment; try the `place-*` shorthands.
- **Dense packing side-effects.** `grid-auto-flow: dense` may reorder visuals; avoid if logical order matters for accessibility.

14 Exercises (Hands-on)

Each exercise targets a specific skill. Create a folder per exercise and an 'index.html' + 'style.css'.

1. 3×3 Number Grid

Make a 3x3 grid with numbered boxes. Center numbers, add gaps. Stretch one box to span two columns.

2. Responsive Gallery

Thumbnails in a fluid grid using `repeat(auto-fit, minmax(160px, 1fr))`. Use `object-fit:cover`.

3. Pricing Table

Three plans (Basic/Standard/Pro). Highlight middle plan with different background. Use Grid for columns; Flexbox inside card headers.

4. Blog Page

Header, sidebar, content, footer using `grid-template-areas`. Collapse sidebar below content on small screens.

5. Dashboard

Place 6 cards in a grid; make the KPI card span two columns on desktop; stack on mobile.

6. Overlap Hero

Create a hero background with a CTA box overlapping bottom-right using line-based placement and `z-index`.

7. Named Lines

Define named grid lines and place sidebar/content using names instead of numeric lines.

8. Product Catalog

Cards arranged via `auto-fill` so gutters remain. Add a filter column on desktop; stack on mobile.

9. Magazine Front

Two-row lead story, sidebar, and a responsive article grid below.

10. Nested Grid Card

Inside a product card, create a nested grid: image, title, description (flexible), and footer actions.

11. Alignment Lab

Experiment with `place-items`, `place-content`, and per-item `align-self`/`justify-self`.

12. Implicit Rows

Start with a 4-column explicit grid; let additional items create implicit rows of fixed height via `grid-auto-rows`.

Further Reading

- MDN Web Docs

End of Chapter.