# RESPONSIVE INTER FACE DESIGN

## CSS Grid Layout Module🧩 What is CSS Grid?

**CSS Grid Layout** is a two-dimensional layout system for the web. Unlike Flexbox (which is one-dimensional), CSS Grid can layout items in **both rows and columns**. It's incredibly powerful for designing **responsive** websites, as it adapts to different screen sizes and orientations seamlessly.

## 🛠️ Key Concepts of CSS Grid

### 1. **Grid Container**

display: grid;

Defines a grid container and enables grid layout.

### 2. **Grid Tracks**

You define rows and columns using grid-template-rows and grid-template-columns.

### 3. **Grid Items**

Children of the grid container automatically become grid items.

### 4. **Grid Lines**

The dividing lines that make up the structure of the grid.

## ✅ Basic Example

html

<div class="grid-container">

<div class="item">1</div>

<div class="item">2</div>

<div class="item">3</div>

</div>

css

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 1rem;

}

.item {

background: #eee;

padding: 20px;

text-align: center;

}

* repeat(3, 1fr) creates 3 equal columns.
* gap adds spacing between rows and columns.

## 📱 Making It Responsive

### Approach 1: **Auto-fit with** minmax()

.grid-container {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

}

* auto-fit automatically fits items into available space.
* minmax(200px, 1fr) means each column will be **at least 200px** but will **grow** if there is space.

🔁 This creates a responsive layout without media queries!

### **Approach 2: Media Queries**

css

.grid-container {

display: grid;

grid-template-columns: 1fr;

}

@media (min-width: 768px) {

.grid-container {

grid-template-columns: repeat(3, 1fr);

}

}

* On small screens: one column layout.
* On medium+ screens: three columns.

## Grid Placement and Spanning

.item-a {

grid-column: 1 / span 2; /\* spans 2 columns \*/

grid-row: 1 / 3; /\* spans 2 rows \*/

}

You can place and size grid items **explicitly**.

## 🧱 Grid Template Areas

.grid-container {

display: grid;

grid-template-areas:

"header header"

"sidebar content"

"footer footer";

grid-template-columns: 200px 1fr;

grid-template-rows: auto 1fr auto;

}

.header { grid-area: header; }

.sidebar { grid-area: sidebar; }

.content { grid-area: content; }

.footer { grid-area: footer; }

### HTML

<div class="grid-container">

<div class="header">Header</div>

<div class="sidebar">Sidebar</div>

<div class="content">Content</div>

<div class="footer">Footer</div>

</div>

* Named areas make your layout readable and manageable.
* You can redefine areas using **media queries** for different screen sizes.

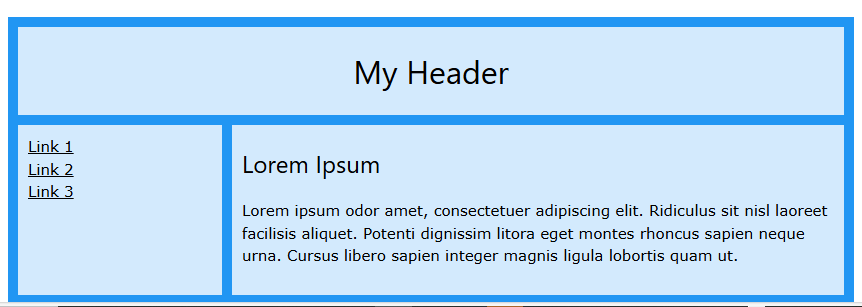
## 🔄 Combining with Flexbox

* Use **Grid for overall page layout**.
* Use **Flexbox inside components** (like navbar, cards, etc.).

## 💡 Best Practices

|  |  |
| --- | --- |
| **Practice** | **Description** |
| Use auto-fit or auto-fill | For fluid layouts |
| Combine with media queries | For more control |
| Use fr units | For scalable column widths |
| Keep your markup simple | Grid reduces nesting |
| Use gap instead of margins | Easier spacing management |

# EXAMPLE 2



## CSS Grid Layout

The Grid Layout Module offers a grid-based layout system, with rows and columns.

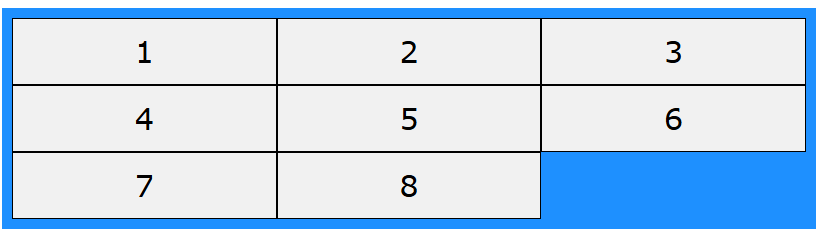
## Grid Container and Grid Items

A grid layout consists of a parent element (the grid container), with one or more grid items.

All direct children of the grid container automatically become grid items.

### **Example**

<div class="container">  
  <div>1</div>  
  <div>2</div>  
  <div>3</div>  
  <div>4</div>  
  <div>5</div>  
  <div>6</div>  
  <div>7</div>  
  <div>8</div>  
</div>



<!DOCTYPE html>

<html>

<head>

<style>

.container {

display: grid;

grid-template-areas:

"header header"

"menu content"

"footer footer";

grid-template-columns: 1fr 3fr;

gap: 5px;

background-color: #2196F3;

padding: 5px;

}

.container > div {

background-color: rgba(255, 255, 255, 0.8);

padding: 10px;

}

.container > div.header {

grid-area: header;

text-align: center;

}

.container > div.menu {

grid-area: menu;

}

.container > div.content {

grid-area: content;

}

.container > div.footer {

grid-area: footer;

}

</style>

</head>

<body>

<h1>CSS Grid Layout</h1>

<p>The Grid Layout Module offers a grid-based layout system, with rows and columns.</p>

<p>The Grid Layout Module makes it easy to design complex and responsive web pages without using floats and positioning:</p>

<div class="container">

<div class="header"><h2>My Header</h2></div>

<div class="menu"><a href="#">Link 1</a><br><a href="#">Link 2</a><br><a href="#">Link 3</a></div>

<div class="content"><h3>Lorem Ipsum</h3><p>Lorem ipsum odor amet, consectetuer adipiscing elit. Ridiculus sit nisl laoreet facilisis aliquet. Potenti dignissim litora eget montes rhoncus sapien neque urna. Cursus libero sapien integer magnis ligula lobortis quam ut.</p></div>

<div class="footer"><h4>Footer</h4></div>

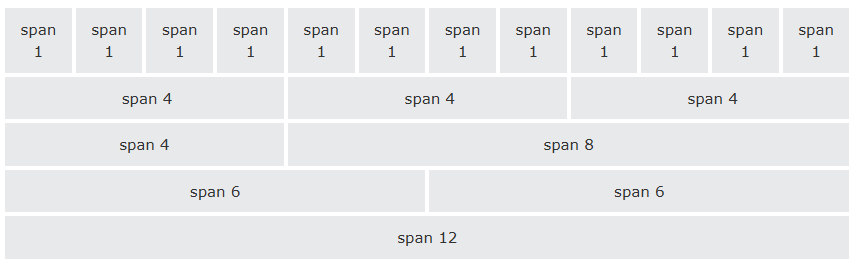
</div>

</body>

</html>

**Bootstrap Grid System**

Bootstrap's grid system allows up to 12 columns across the page.

If you do not want to use all 12 column individually, you can group the columns together to create wider columns

Bootstrap's grid system is responsive, and the columns will re-arrange depending on the screen size: On a big screen it might look better with the content organized in three columns, but on a small screen it would be better if the content items were stacked on top of each other.

**Tip:** Remember that grid columns should add up to twelve for a row. More than that, columns will stack no matter the viewport.

Grid Classes

The Bootstrap grid system has four classes:

* xs (for phones - screens less than 768px wide)
* sm (for tablets - screens equal to or greater than 768px wide)
* md (for small laptops - screens equal to or greater than 992px wide)
* lg (for laptops and desktops - screens equal to or greater than 1200px wide)

The classes above can be combined to create more dynamic and flexible layouts.

**Tip:** Each class scales up, so if you wish to set the same widths for xs and sm, you only need to specify xs.

Grid System Rules

Some Bootstrap grid system rules:

* Rows must be placed within a .container (fixed-width) or .container-fluid (full-width) for proper alignment and padding
* Use rows to create horizontal groups of columns
* Content should be placed within columns, and only columns may be immediate children of rows
* Predefined classes like .row and .col-sm-4 are available for quickly making grid layouts
* Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on .rows
* Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three .col-sm-4
* Column widths are in percentage, so they are always fluid and sized relative to their parent element

**Basic Structure of a Bootstrap Grid**

The following is a basic structure of a Bootstrap grid:

<div class="container">  
  <div class="row">  
    <div class="col-\*-\*"></div>  
    <div class="col-\*-\*"></div>  
  </div>  
  <div class="row">  
    <div class="col-\*-\*"></div>  
    <div class="col-\*-\*"></div>  
    <div class="col-\*-\*"></div>  
  </div>  
  <div class="row">  
    ...  
  </div>  
</div>

So, to create the layout you want, create a container (<div class="container">). Next, create a row (<div class="row">). Then, add the desired number of columns (tags with appropriate. col-\*-\* classes). Note that numbers in .col-\*-\* should always add up to 12 for each row.

## Grid Options

The following table summarizes how the Bootstrap grid system works across multiple devices:

