CSS GRID

Tartalom

[Introduction 2](#_Toc152584966)

[In the upcoming period, we will delve deeply into the study of CSS Grid, which enables us to create real responsive two-dimensional grid layouts using Pure CSS. Our curriculum will cover the following: 2](#_Toc152584967)

[Specification of the CSS Grid Layout module 2](#_Toc152584968)

[Designing and implementing CSS grid layouts in a live environment 2](#_Toc152584969)

[Special layouts with CSS Grid (including full layouts, multi-column layouts, card layouts, nested layouts, and even interactive canvas menus) 2](#_Toc152584970)

[The CSS Grid Layout module changes almost everything about the web layouts you've learned so far and opens up new possibilities for designing layouts. CSS Grid is a fundamental CSS layout module, similar to block, inline, flex, table, and so on, which you have already become acquainted with. This prior knowledge is necessary to understand how CSS Grid works. 2](#_Toc152584971)

[1. Basics of CSS GRID 2](#_Toc152584972)

[2. Create a Grid 4](#_Toc152584973)

[3. Grid Lines and Units 9](#_Toc152584974)

# Introduction

# In the upcoming period, we will delve deeply into the study of CSS Grid, which enables us to create real responsive two-dimensional grid layouts using Pure CSS. Our curriculum will cover the following:

# Specification of the CSS Grid Layout module

# Designing and implementing CSS grid layouts in a live environment

# Special layouts with CSS Grid (including full layouts, multi-column layouts, card layouts, nested layouts, and even interactive canvas menus)

# The CSS Grid Layout module changes almost everything about the web layouts you've learned so far and opens up new possibilities for designing layouts. CSS Grid is a fundamental CSS layout module, similar to block, inline, flex, table, and so on, which you have already become acquainted with. This prior knowledge is necessary to understand how CSS Grid works.

# Basics of CSS GRID

***A Grid Container*** is any element whose display property is set to grid, meaning when we set the container's property and value to display: grid. In a single view or document in the browser, you can have as many or as few grid containers as you need, and you can nest grid containers within other grid containers.

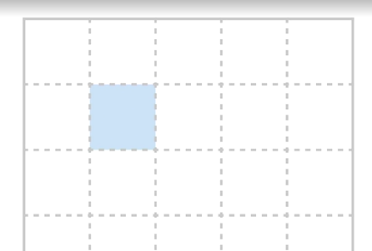
The immediate direct children of the grid container become grid items. They automatically go onto the grid unless specified otherwise, and you can precisely position each grid item within the grid, placing it exactly where you want it to appear.

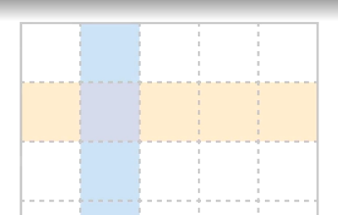
It's important to note that, similar to Flexbox, only the immediate children of the grid container go onto the grid. The descendants of these children are not affected by the grid.

The grid container has four grid lines per box, essentially representing the outer edges of the grid. Additional rows and columns are usually added with row and column tracks to segment the grid into cells.

The grid lines are referenced by a number, starting and ending with the outer edges of the grid container. Thus, the left and top edges are typically considered the 1st row, and each subsequent row is assigned an incrementing number, as seen in the illustration above.

CSS Grid respects the current document's text direction or even the direction of the current element. When moving from left to right, the left outer edge is considered the 1st row, and the last row is the right outer edge. In settings where the direction is right-to-left—such as in Arabic text—the numbering is reversed, starting with the right outer edge as the 1st row and ending with the left outer edge.

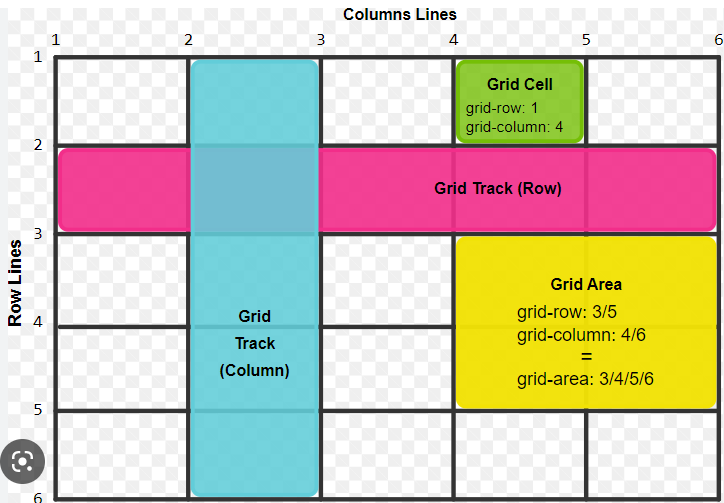
You can reference a grid line by its number, or you can assign custom names to grid lines and refer to them using these custom names..

A **grid cell** is the space between two adjacent horizontal and vertical grid lines, where a grid column and a grid row intersect. It's roughly equivalent to a cell in a table in applications like Excel.

A **grid track** is the space between two or more adjacent horizontal or vertical grid lines. Row tracks are horizontal, while column tracks are vertical.

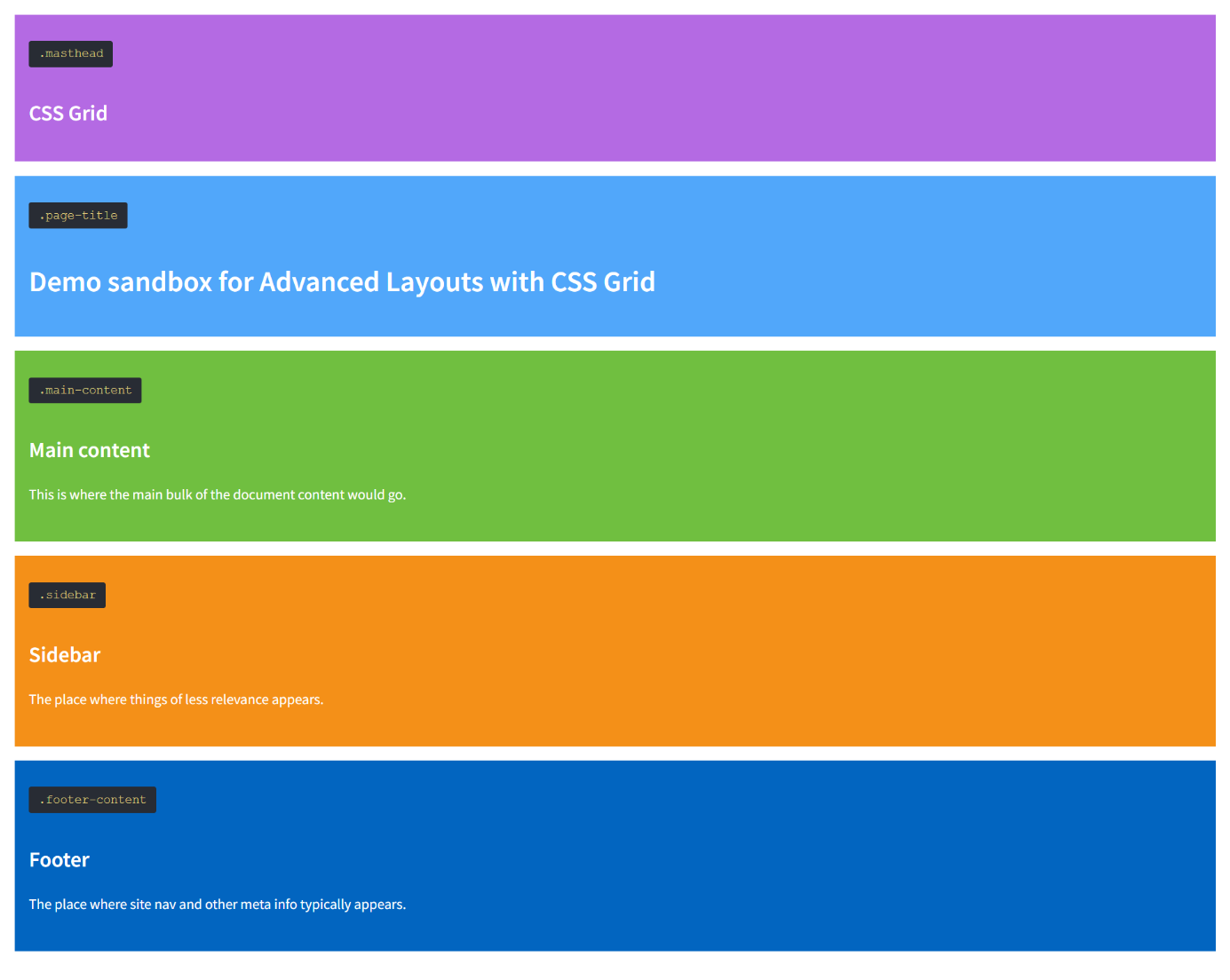
A **grid area** covers one or more cells, and its purpose is to define a region within the grid that can be referenced when placing grid items. You can add space between individual grid tracks by introducing a grid gap. The grid gap creates uniform distances between the tracks, and you can specify gaps for both column and row tracks independently.

It's important to note that when using a grid gap, you essentially create empty space between the individual tracks, meaning that any color or element behind the grid will be visible through this gap.



# Create a Grid

The following layout will be created in the first step:



Create an empty folder and an index.html document. The following HTML tags should be placed in the body section of the document:

<body class="site">

<a class="skip-link screen-reader-text" href="#content">Skip to content</a>

<header class="masthead">

<code class="identifier"><pre>.masthead</pre></code>

<h2>CSS Grid</h2>

</header><!-- .masthead -->

<div class="page-title">

<code class="identifier"><pre>.page-title</pre></code>

<h1>Demo sandbox for Advanced Layouts with CSS Grid</h1>

</div><!-- .page-title -->

<main id="content" class="main-content">

<code class="identifier"><pre>.main-content</pre></code>

<h2>Main content</h2>

<p>This is where the main bulk of the document content would go.</p>

</main><!-- #content .main-content -->

<aside class="sidebar">

<code class="identifier"><pre>.sidebar</pre></code>

<h2>Sidebar</h2>

<p>The place where things of less relevance appears.</p>

</aside><!-- .sidebar -->

<footer class="footer-content">

<code class="identifier"><pre>.footer-content</pre></code>

<h2>Footer</h2>

<p><span class="bg">The place where site nav and other meta info typically appears.</span></p>

</footer><!-- .footer-content -->

</body>

Másold be a kódot a dokumentumodba.

Hozz létre egy css fájlt és csatold az oldalhoz.

A fejlécbe húzd be a következő betűtípust:

<link href="https://fonts.googleapis.com/css?family=PT+Serif:400,400i,700,700i|Source+Sans+Pro:400,400i,600,600i" rel="stylesheet">

If we look at the HTML code of this document, we can see that it has a body with a "site" class, containing a series of first-level descendants with classes like "skip-link," "masthead," "page-title," and "maincontent." In the browser, these elements appear sequentially as individual blocks.

Now, we will experiment with different layouts for these boxes, or blocks, using CSS Grid.

The first step is to declare a grid container, and in this case, the container is the body element with the "site" class because it contains all the other elements. Let's create two external CSS files for this: one named "grid.css" and the other "main.css." Attach "main.css" first and then "grid.css" to your index.html page. Let's format "main.css" together first.

Since you already know how to style individual boxes, color them, and format text, I'll provide the content of "main.css" so that we can save some time and focus on the new material::

.screen-reader-text {

clip: rect(1px, 1px, 1px, 1px);

position: absolute !important;

height: 1px;

width: 1px;

overflow: hidden;

}

.screen-reader-text:hover,.screen-reader-text:active,.screen-reader-text:focus {

background-color: #f1f1f1;

border-radius: 3px;

box-shadow: 0 0 2px 2px rgba(0, 0, 0, 0.6);

clip: auto !important;

color: #21759b;

display: block;

font-size: 14px;

font-weight: bold;

height: auto;

left: 5px;

line-height: normal;

padding: 15px 23px 14px;

text-decoration: none;

top: 5px;

width: auto;

z-index: 100000;

}

/\*--------------------------------------------------------------

Format of the page

--------------------------------------------------------------\*/

body {

margin: 0;

}

body,button,input,select,textarea {

font-family: 'Source Sans Pro', 'Helvetica', 'Arial', sans-serif;

font-size: 18px;

line-height: 1.5;

}

h1,h2,h3,h4,h5,h6 {

clear: both;

}

p {

margin-bottom: 1.5em;

}

b,

strong {

font-weight: bold;

}

dfn,cite,em,i {

font-style: italic;

}

blockquote {

margin: 0 1.5em;

}

address {

margin: 0 0 1.5em;

}

pre {

display: inline-block;

font-family: "Courier 10 Pitch", Courier, monospace;

color: #ddca7e;

background: #282c34;

padding: .5em .8em;

border-radius: .2em;

}

code,kbd,tt,var {

font: 15px Monaco, Consolas, "Andale Mono", "DejaVu Sans Mono", monospace;

}

abbr,acronym {

border-bottom: 1px dotted #666;

cursor: help;

}

mark,ins {

background: #fff9c0;

text-decoration: none;

}

sup,sub {

font-size: 75%;

height: 0;

line-height: 0;

position: relative;

vertical-align: baseline;

}

sup {

bottom: 1ex;

}

sub {

top: .5ex;

}

small {

font-size: 75%;

}

big {

font-size: 125%;

}

/\* Kisebb kijelzőre definiálva \*/

@media screen and (min-width: 700px) {

.two-column {

column-count: 2;

column-gap: 2.4em;

}

}

/\* Global buttons \*/

.content-button {

padding: .5em 1em;

border: 3px solid #B51C35;

/\*border-radius: 10px;\*/

font-weight: 600;

}

a.content-button {

color: black;

text-decoration: none;

}

a.content-button:focus,a.content-button:hover {

background: #B51C35;

color: #FFFCED;

}

/\*--------------------------------------------------------------

Basic style

--------------------------------------------------------------\*/

.site {

margin: 1em;

}

.masthead,.page-title,.main-content,.sidebar,.footer-content {

margin-bottom: 1em;

padding: 1em;

color: white;

}

.masthead {

background-color: #b46ae3;

}

.page-title {

background-color: #51a7fa;

}

.main-content {

background-color: #70bf40;

}

.sidebar {

background-color: #f49018;

}

.footer-content {

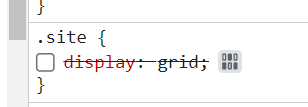
background-color: #0265c0;

}

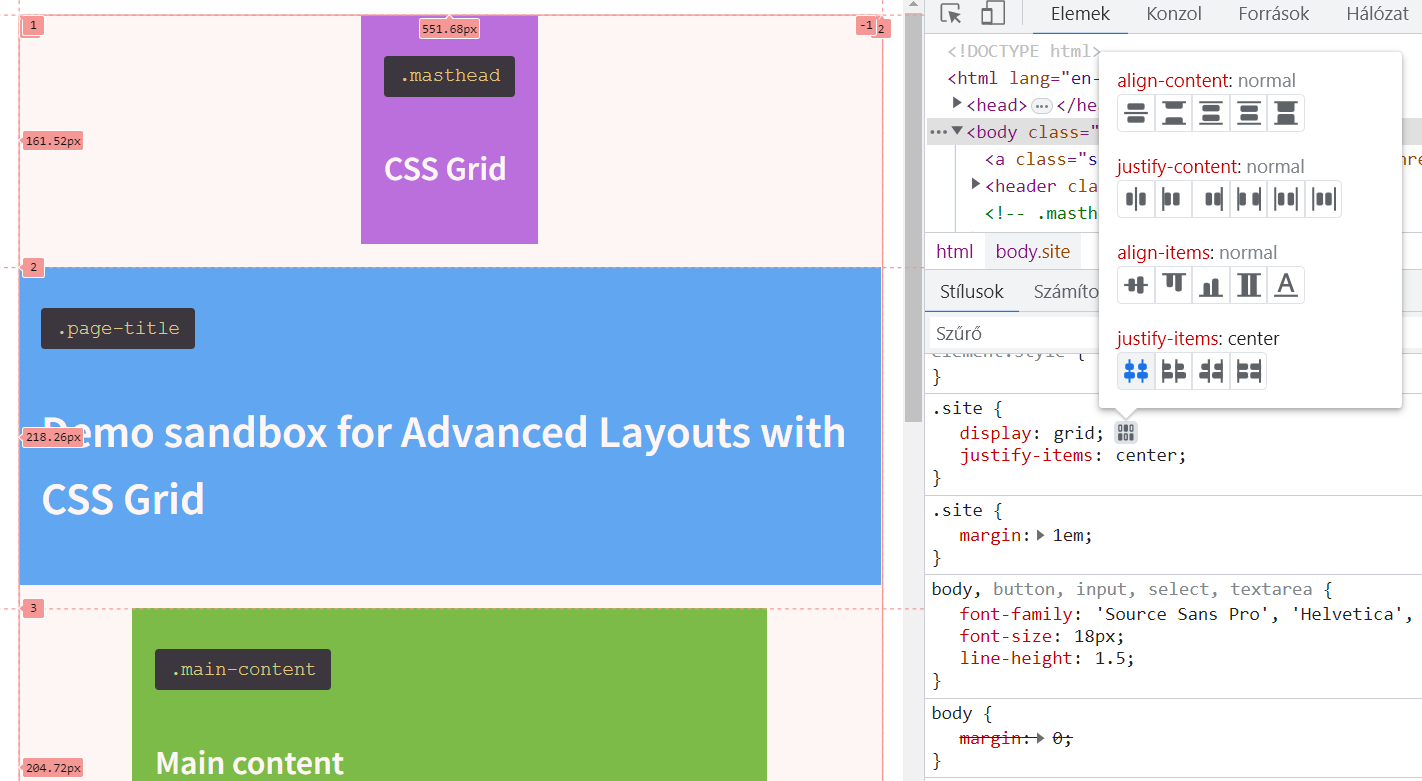
In the "grid.css," we select a single selector for the body, specifically the class associated with the document body, which is indicated by ".site." The property is "display," and its value is "grid."

If you now open the developer tools and select the body, you will see a small grid pattern next to it. Click on it, and it will display the grid view on your page.

Now, if you look at the grid, you will see that there is an outline representing the grid itself and that each element is a grid block. In fact, these are placed within individual grid cells. Currently, these grid cells happen to be arranged vertically and positioned one after the other. So, practically, there is no difference between the state before setting up the grid and the current state.☺

 Now we can click on the Layout tab on the right side, and you will see that there is an Overlay Grid underneath the body.site. Currently, this is enabled. You can toggle it on and off.

Within this tool, you can now choose exactly what you want to visualize. So, you can extend the grid lines indefinitely. Here, you can see that the grid lines extend beyond the grid container itself.

 Here, you can now see those grid lines I was talking about. It can display the row numbers, here's the first row, the second, the third, and so on. Currently, we don't have any columns yet, so it's just the first and second rows for the columns. Finally, we can display the grid areas, but we don't have any grid areas yet, so there's no point in turning this on. This tool also allows us to see which grid areas we are currently focusing on by simply hovering over the display.

From now on, for the exercises, we will use the same folder, but we will make changes in the "grid.css" file..

# Grid Lines and Units

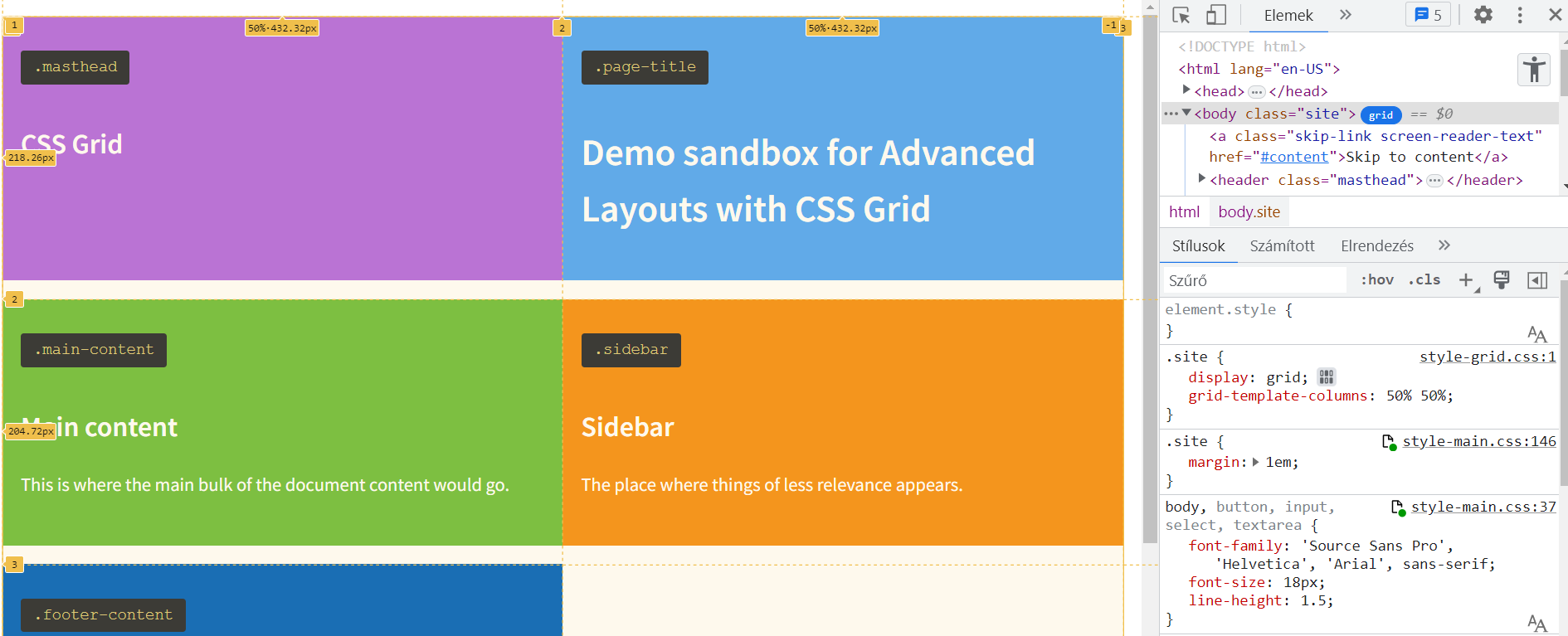
By defining the grid container, we can start declaring grid lines to create columns and rows. This is done using the properties of grid-template-columns and grid-template-rows.

Each of these properties takes space-separated lists of values that set the width or height of the columns or rows by declaring the width or height of the space between one row and the next.

Currently, in the browser, the grid has one column that spans the entire width of the grid. The first column grid line is at the left edge, and the second column grid line is at the right edge.

Let's add a new property to the .site class: grid-template-columns: 50% 50%.

Save it, and in the browser, you will see that an additional grid line has been created. So, there is one grid line at the left edge, the second grid line is now in the middle, and the third grid line is at the right edge.

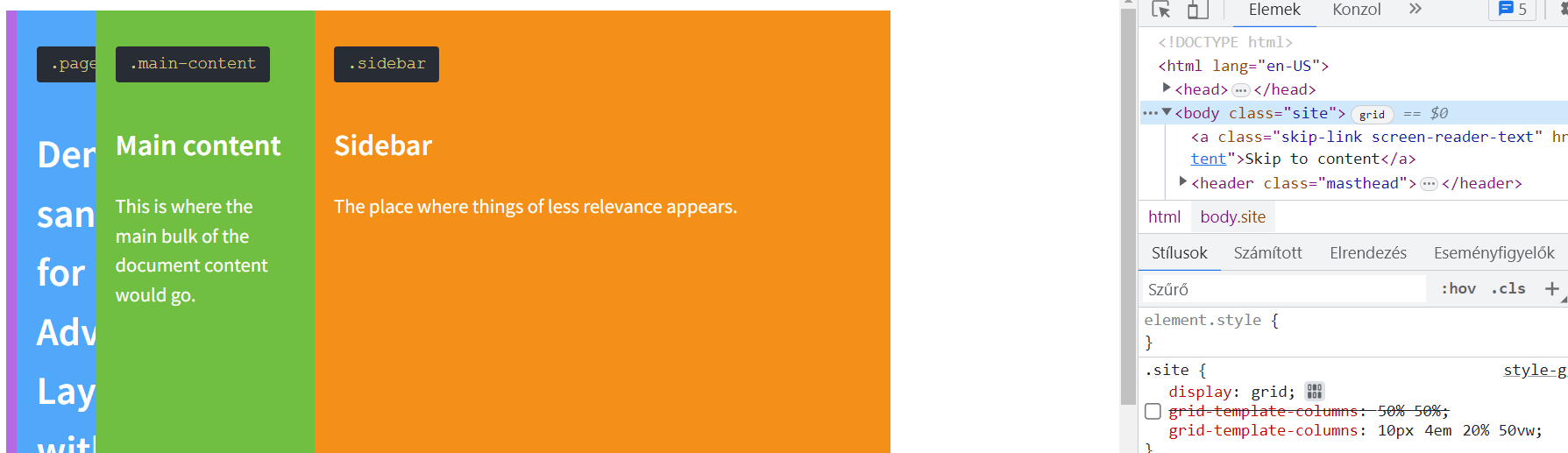


*7. pic*

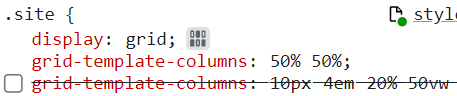
The property we added instructs to find the first grid line, which is the left edge, then place a second grid line at 50% of the total width of the grid area, and place the last grid line after the remaining 50%. As a result, we get two columns with equal widths within our grid. The default width of created rows or columns is automatic, meaning it is as wide as the content needs within the available space of the grid.

If you want a column or row to expand or shrink based on its content, you can specify column widths and row heights using a mix of different units. For example:

grid-template-columns: 10px 4em 20% 50vw



Exactly, as shown in the example above, you can mix pixel widths, em units, percentages, viewport width (VW), and anything else in the same declaration. So, if you want to create a very peculiar grid, you could say something like 10 pixels, 4 em units, 20%, and 50 VW.

 Yes, you can easily toggle properties on and off in the developer tools to test what happens:

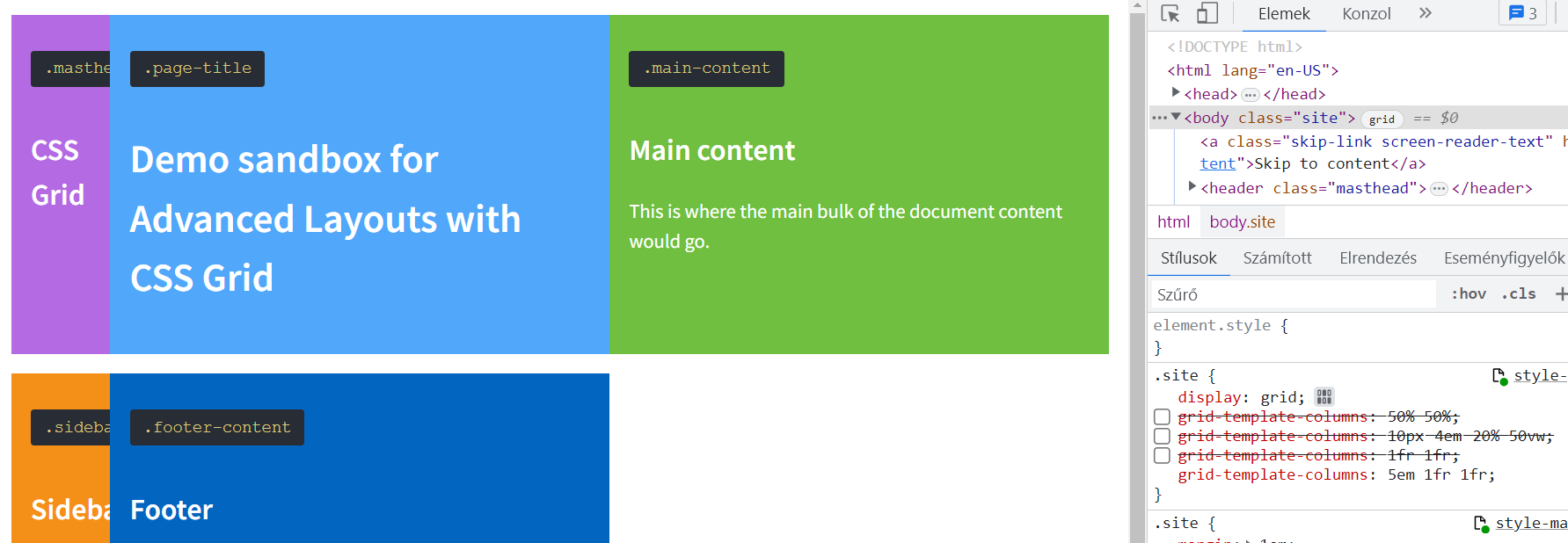
CSS-Grid has two additional measurement units and tools that you can use: the Fraction (FR) Unit and the minmax function. Using the fraction unit, indicated as **FR,** is quite simple. In our example, instead of saying we want 50% of the area, we just divide the grid into two equal fractions:

grid-template-columns:1fr 1fr;

If you look in the browser, we will get the same result as when we divided the grid into two equal 50% fractions. (*7.pic*)

Exactly, if you want to create another column with a fixed width, for example, 5 em, the remaining space will still be evenly distributed between the two FR units.

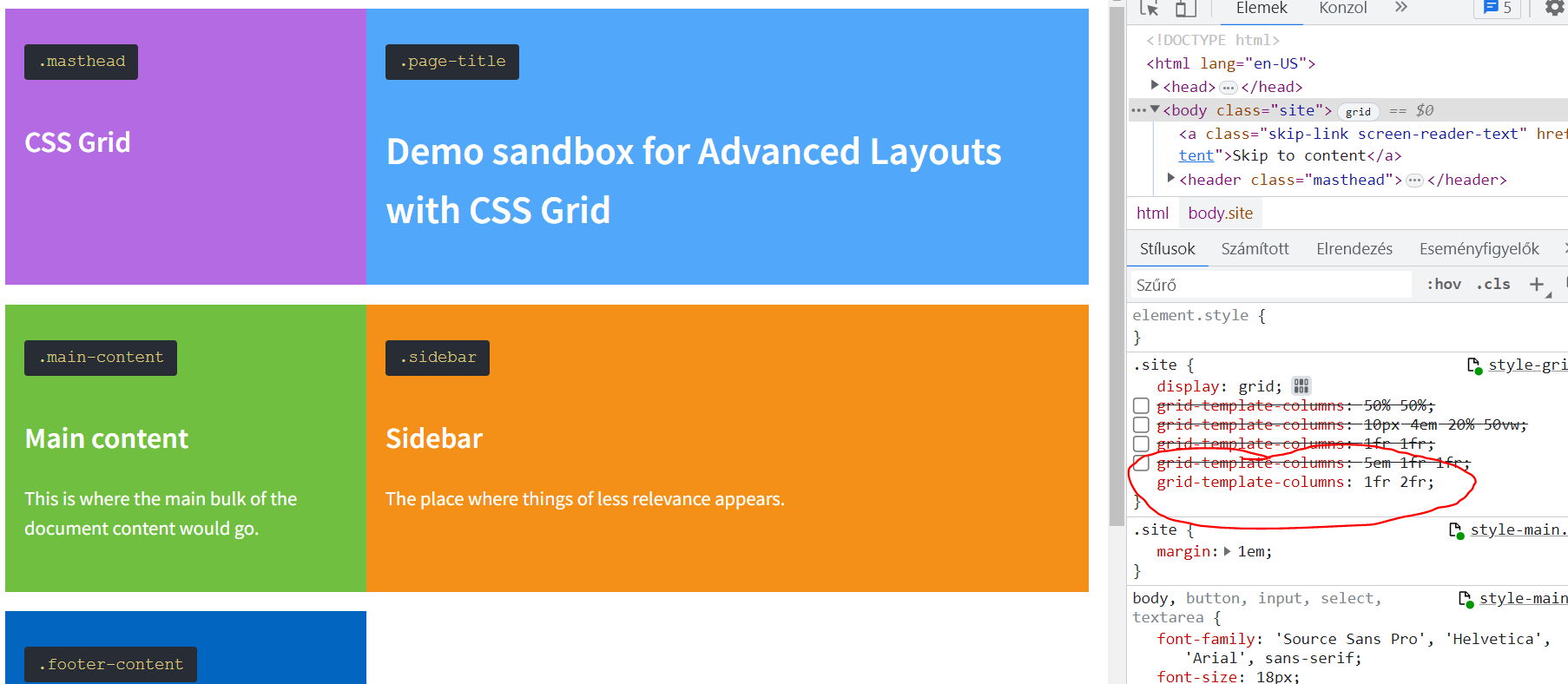
grid-template-columns: 5em 1fr 1fr;



The FR unit allows us to work with fractions. So currently, if you have one FR and one FR, the two columns are of equal width.

But if you set 1 FR and 2 FR, it divides it into three parts.:

grid-template-columns: 1fr 2fr;

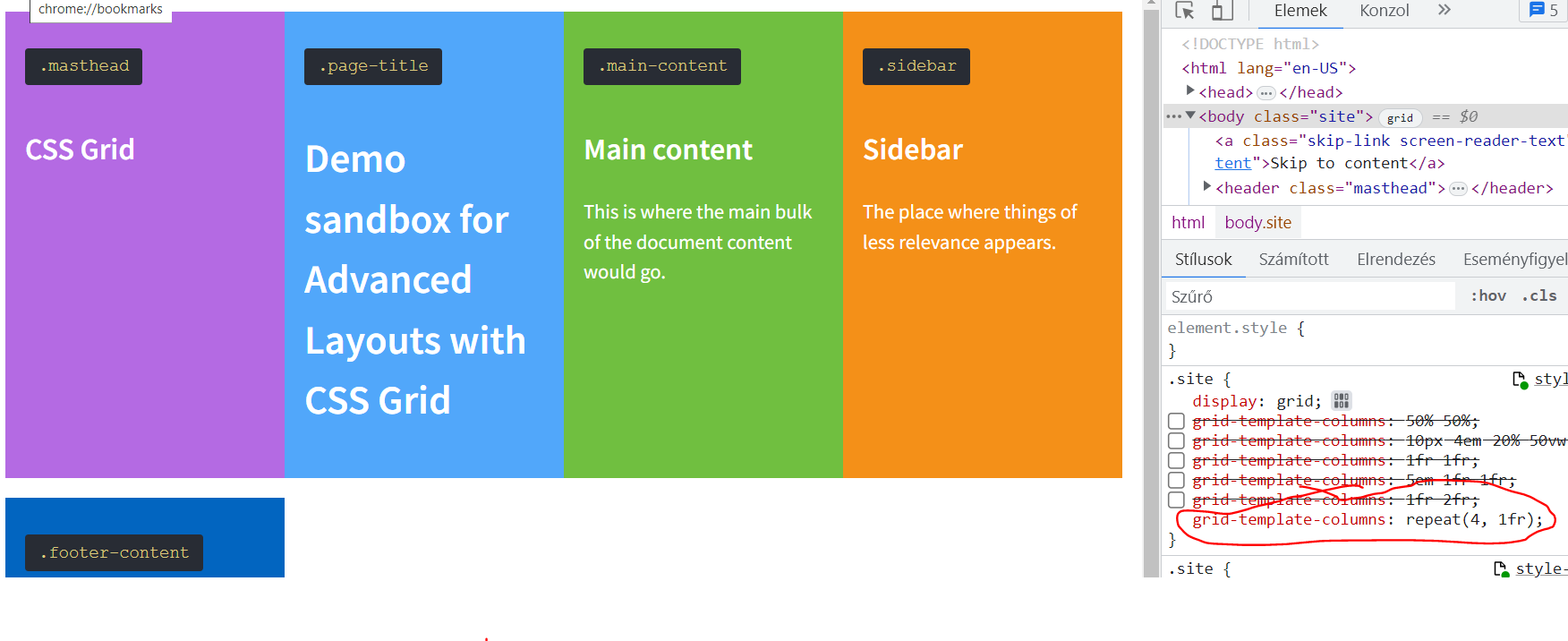


So, in the end, we get something more precise than if we were to try to specify values like 33.3333333% three times.

The minmax() function is a bit more advanced than this. It allows you to declare the minimum and maximum width of a column or row. This is particularly useful and sensitive for design because you can specify that certain columns can never be narrower than 20 em and never wider than 30 em. It's similar to setting a minimum and maximum width but in the context of columns and rows. This is important for us because it allows us to create truly responsive layouts using CSS Grid.

If you need multiple similar columns, there is a shortcut using the repeat() notation. Let's say you want four columns, each with 1 FR. Instead of typing 1 FR, 1 FR, 1 FR, 1 FR, you can simply write it like this:

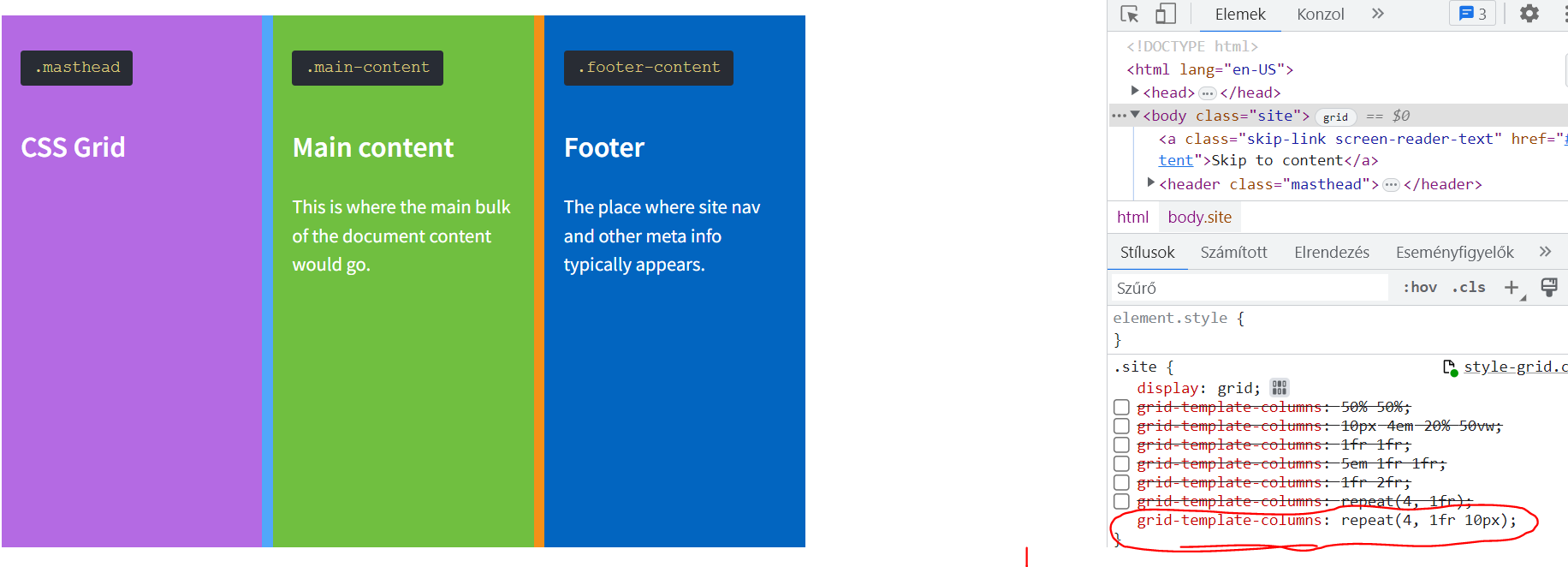
grid-template-columns: repeat(4, 1fr);



We can also use more advanced combinations, for example:

grid-template-columns: repeat(4, 1fr 10px);

Syntax is crucial—where you place commas and where you don't! So, repeat the four, separated by commas, 1 FR 10 pixels, and notice there are no commas between these! The result actually gives you eight columns, alternating between 1 FR and 10 pixels in width.



We can also provide a mixed setting, fixed for one grid track and automatic for the rest:

grid-template-columns: 2fr repeat(2, 1fr);

