CSS Grid for Designers

How a new technology is changing layout on the web

Johna Paolino

Jan 3 · 10 min read

Illustration by Dominic Kesterton

For years, designers have been using grids to bring order to pages. Grids as a design tool are associated with the Swiss who formalized it as a way of thinking about layout in the 1940s, according to Beth Tondreau in the book “Layout Essentials”. As people started designing for the Internet, grid systems were carried from the printed page to the digital one.

During this time, CSS (the code that controls the style of elements in your browser) was limited in terms of layout capabilities. My teammate, Natalya Shelburne, an engineer at The New York Times, equates it to trying to create designs using the tooling of Microsoft Word. As a workaround to these limitations, a number of layout frameworks were developed to make working with lay-

out easier. In 2011, Twitter released Bootstrap, one of the more popular layout solutions. Bootstrap did a bunch of calculations behind the scenes, so that developers could use simplified code to implement layouts on a 12-column grid.

Grids on the web were not simply design guidelines for layout, but actual code that both limited and executed the placement of elements across viewports and breakpoints.

Fast forward six years to the release of CSS Grid in early 2017. This technology removes a lot of the limitations that existed in CSS to date. But CSS Grid is not just a tool for front-end developers; designers can now think about web layout in new ways.

CSS Grid makes it easy to create **grid tracks** using CSS—grid tracks are just a fancy way of saying columns and rows. The 12-column grid popularized by Bootstrap was a smart solution at the time, but CSS Grid gives us many more layout options.

The 12-column grid can support a variety of design layouts like single column, two-column and three-column.

CSS Grid allows us to quickly create custom grids for our projects. This means we don't need to start with 12 columns—we can have five or eight.

> *This technology advancement is so important for designers to understand, because it means we can think about the right grid for the demands of our content.*

We can create grids that better control the placement of elements both vertically and horizontally. All of this brings classic graphic design and art direction approaches back to web design.

• • •

How to get started with CSS Grid

The language of CSS Grid is incredibly straightforward, so there's no need to learn cryptic classnames that were popularized with Bootstrap, like `col-sm-8`. There are two main steps to set up a page using CSS Grid: define the grid tracks and then place elements in those areas.

Defining the Grid Tracks

To set up the grid, we'll create a parent element that will contain all of the HTML elements in the grid. This parent element is typically a `<div>` with a classname of `grid-container` or `container`. In order to identify this container as the grid, we use `display:grid` in the CSS.

Now that the grid container is on the page, we want to start dividing the grid up into columns. To define columns we'll use `grid-template-columns`, and to define the gaps between columns we'll use `grid-column-gap`. The number of columns on the page is controlled by the number of values that are defined in `grid-template-columns` (we can see in the example below there are three values that translate to three columns).

In more traditional graphic design, column width is determined through a calculation of paper size and type setting. Since web design is never constrained to a specific width, sometimes a column width set to percent is the best choice. CSS Grid accepts

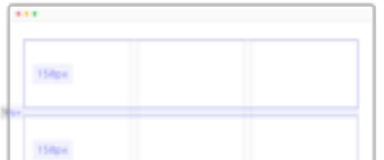
percentages, pixels and fractional units when defining column widths.

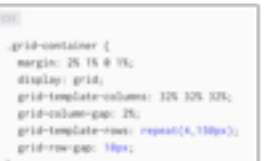
You're probably familiar with pixel and percentage values, but I want to take a moment to quickly endorse **fractional units**, which are new to CSS Grid. Fractional Units, or frs, deal with gnarly math fractions by perfectly dividing the remaining space available. For example, if we wanted a grid to have three columns with no gap, the percent would have a rounded decimal value like: `grid-template-columns: 33.33% 33.33% 33.33%`. Fractional units take the guesswork out of declarations, and instead allow us to write: `grid-template-columns: 1fr 1fr 1fr`. Adding columns or adjusting ratios is simple. Just increase or decrease the numbers, like so: `grid-template-columns: 1fr 2fr 1fr`.

CSS Grid also accepts `auto` for column width, which is a hugely powerful tool. `auto` will expand the column to the maximum width available, as defined by the surrounding columns. For example, if `grid-`

`template-columns` is `100px auto 100px` and the page is 500 pixels wide, the auto section will take up the remaining 300 pixels. If we want finer control over the range, we can also use `minmax()` which limits the column width to the ranges we specify, such as `minmax(550px, 800px)`.

We've gone over columns, but CSS Grid also allows row definition. Rows can be great for finer control over the vertical rhythm of the page, and are created in the exact same way as `grid-template-columns` by using `grid-template-rows` and `grid-row-gap`.





In the repeat() function above, the first number (in this case, 4) is how many times the second number (150px) should be repeated.

Placing Elements in the Grid

O.K., grids are great, but they are invisible unless there is content in them. To ensure grids appear on the page, we need to place elements in each grid track. Grid will *implicitly* place items. But we want to learn how to *explicitly* place items for clearer control. There are two approaches to this.

One approach is to identify the grid lines in which an element should live. Since tracks are the space between lines, this approach specifies the tracks the element will live in. For example, if we wanted to place an element in the third column track, we would tell CSS Grid that the element starts at the third grid line and ends at the last grid line. This would look something like:

This approach is important to learn, but it is most useful if we are trying to put multiple elements in the same grid track.

The other approach, which I prefer, specifies placement using `grid-template-areas`. This allows us to name specific areas where elements should render in the grid, but most importantly it lets us keep all our layout code in one neat place.

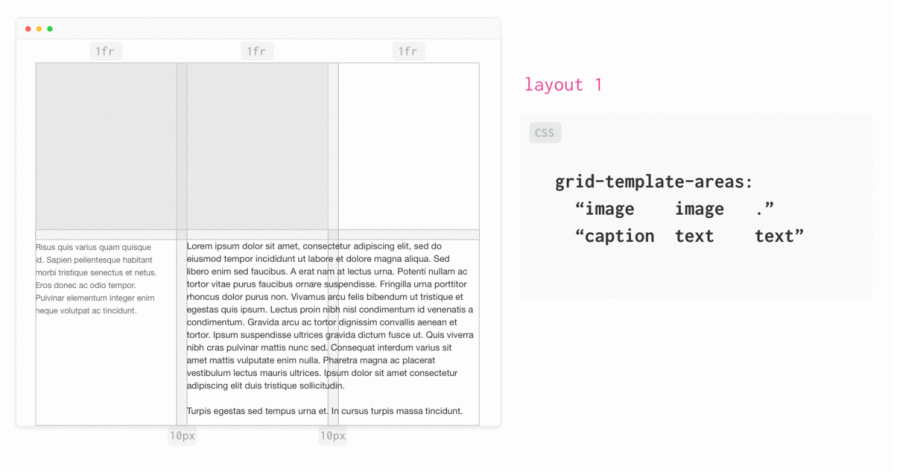
To set this up, each element first needs a `grid-area` declaration in its CSS, which then can be referenced in the parent container declaration. Once these elements are named, we can use the names to arrange the layout visually in the parent `grid-container`.

In the example below, the image is as-

signed to the first column and second column by using the syntax, `"image image ."`. Repeating `image` tells CSS to span the image the first and second column. The `.` tells CSS to keep that area blank. Each new line in the `grid-template-areas` declaration correlates with a new row in the design. In the second line, `caption text text` tells CSS that the second row should have the caption in the first column, and the text element spanning the second and third columns.

What's even more fun with this approach is that it makes it easy to quickly rearrange our layout without needing to rename elements or move the HTML around. We only need to switch placement of the names in the `grid-template-area` declaration to

dramatically change the layout of elements.



These techniques all work responsively, and this is where CSS Grid shines. To define a layout for a smaller breakpoint, we just need to redefine our grid and reorganize the properties in `grid-template-areas`.

The smaller breakpoint at right defines a new grid with two columns.

Placing items in the grid using `grid-template-areas` allows us to actually see our

layout spelled out in text. To me, it feels like designer ASCII art and I love it.

I don't want to understate how exciting this is; gone are the days of `col-lg-2` and `col-sm-12`. We can reorganize our layout at *any* breakpoint, and `grid-template-areas` means our design is visible in the code, across all breakpoints.

Next Level CSS Grid

Granular Placement and Control

Once elements have been placed in a grid track or area, we can further customize their vertical and horizontal alignment—similar to how alignment control work in design software.

`Align-self` and `justify-self` can be applied to the child elements like so:

Grids on Grids on Grids

CSS Grid can be used more than once on a page. Since grids are just containers with elements, there is no reason why they can't be nested inside each other or stacked on top of one another. The next iteration planned for CSS Grid is adding Subgrids for more powerful nesting features.

In the meantime, the current specification for nesting and stacking grids is still great for a variety of instances. For example, if one moment on a page requires very different placement of elements than the rest of the page, we might use a nested grid. This allows us to be precise with layout in this one moment, without overcomplicating the main grid on the page. The technique is especially useful if the main grid is still in progress, because we won't lose nested designs by making changes to the main grid (like adding another column.)

The left image shows one grid container handling all the layout decisions, where the right image shows how a nested grid could be used to handle a more complicated layout moment on the page. Each are valid approaches, but sometimes a nested grid can be cleaner both conceptually and in the code.

Similarly, we can stack grids. This is a great approach if the design is trying to achieve a visual break from the grid that recurs throughout the page.

The left image shows one grid handling all the layout decisions, where the right image shows how a stacked grid could handle a very different layout moment on the page. Each are valid approaches, but sometimes a stacked grid can be cleaner both conceptually and in the code.

. . .

TLDR; Why CSS Grid is great for designers

Wow. That was a lot. Hopefully this guide gave you a pretty comprehensive look at how you can use CSS Grid. Here are the highlights:

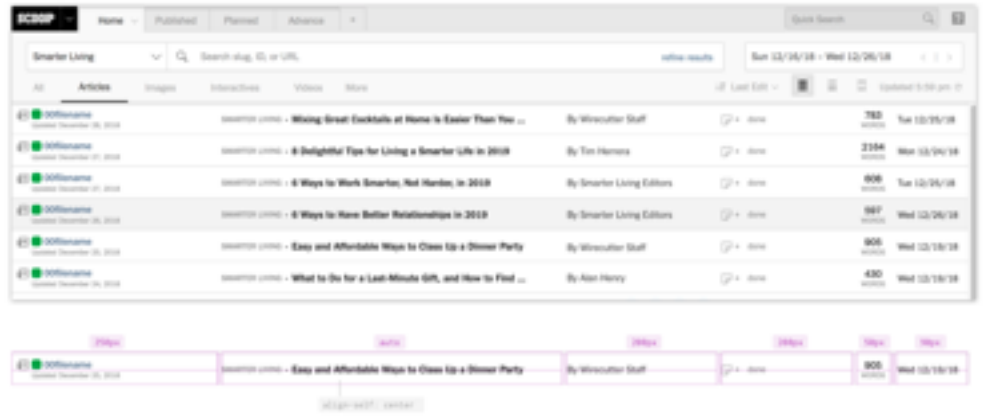
- **No more 12-column grids!** CSS Grid gives you complete control over how many columns and rows your design needs. You have fine grain control over width down to a pixel, but also flexible options including `minmax()` and `auto`.

- **CSS Grid makes it easy to define new grids at any breakpoint**, meaning you can easily redefine grids and rearrange elements across breakpoints. Items on mobile no longer have to stack in the order they are arranged on desktop (unless you them want to).
- **`Grid-template-areas` keeps all your layout code in one place**, with the added bonus of making your layout decisions feel like cool ASCII art.
- **CSS Grid lets the content determine the page layout** instead of forcing the content to fit into an existing grid structure. This follows Brad Frost's words of advice, "let content determine breakpoints."

• • •

These layout advancements are great for all types of design. I recently used CSS Grid for a technical search interface for The New York Times's Content Management System, Scoop. Each search result in this interface is

a grid container, using an `auto` column for the headline cell.



Each search result is its own grid container with two rows and six columns.

I've recently moved onto a team with more of an editorial focus and am looking forward to finding out how CSS Grid fits in to that work. CSS Grid has personally made me love HTML and CSS prototyping again, and I hope it does the same for you.

Helpful Resources

There are a lot of details that I gave short shrift, but below are some resources that go into more detail. Happy reading!

- CSS Grid Garden

- A Complete Guide to CSS Grid from CSS Tricks
- CSS Grid—The Beginner's Guide
- CSS Grid Layout from MDN
- Youtube: Layout Land

People:

- Jen Simmons
- Rachel Andrew
- Wes Bos

CSS | Css Grid | Design | Front End Development | UI Design

5K claps

14

Johna Paolino

Product design @nytimes. // TODO share work & code with more people

Follow

Times Open

Sharing our stories of making great digital products at The New York Times.

Follow

More from Times Open

How We Prepared New York Times Engineering for the Midterm Elections

The Times Open Team

Feb 15 · 8 min read

55

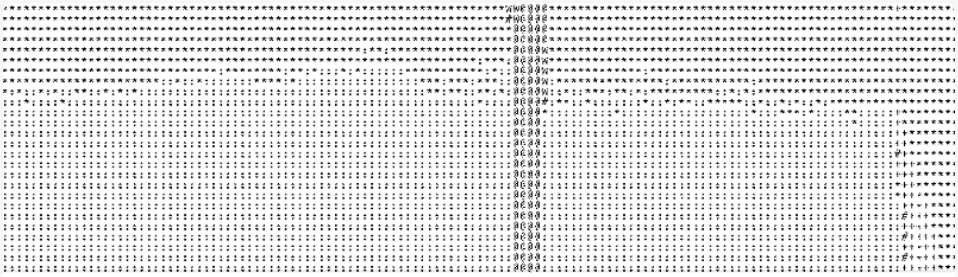
More from Times Open

No Code? No Problem—Writing Tests in Plain English

The Times Open Team

Feb 1 · 4 min read

338



More from Times Open

Building a Text Editor for a Digital-First Newsroom

Sophia Ciocca

Apr 12, 2018 · 10 min read

13.4K

Responses

Applause from Johna Paolino (author)

Chris Raymond

Jan 5

Excellent explanation, really clear images in context. Thanks, Johna.

9

Conversation between Edgar Fleming and Johna

Paolino.

Edgar Fleming
Jan 5

Well done. A complete & refined writeup with inclusive reach for both laymen & the seasoned professional. Thank you.

15 1 response

Johna Paolino
Jan 7

Thank you so much! That's all I could've hoped for.

Show all responses