This article was updated in July, 2018 to reflect improved browser support for CSS Grid Layout.

As web applications become more and more complex, we need a more natural way to do advanced layouts easily without hacky solutions that use floats and other less burdensome techniques. An exciting new solution for creating layouts comes with the CSS Grid Layout Module (https://drafts.csswg.org/css-grid/)**.**

In this introductory tutorial, I'll introduce you to this relatively new CSS feature, I'll discuss the current browser support, and I'll show you using some examples how the CSS Grid Layout Module works.

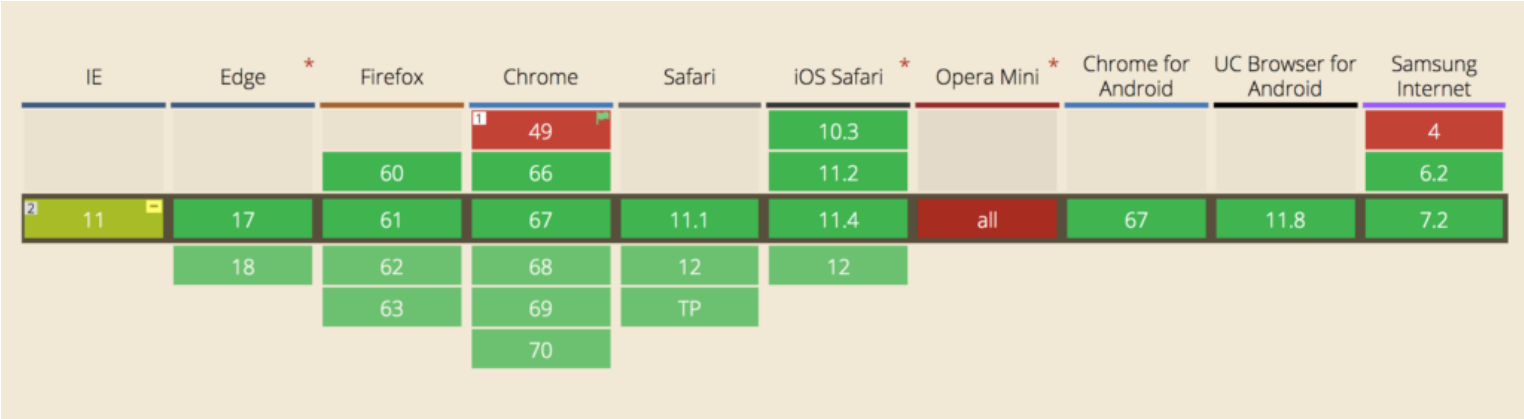
What is the CSS Grid Layout Module?

The core idea behind the Grid Layout is to divide a web page into columns and rows, along with the ability to position and size the building block elements based on the rows and columns we have created in terms of size, position, and layer.

The grid also gives us a flexible way to change the position of elements with only CSS without any change to the HTML. This can be used with media queries to alter the layout at different breakpoints.

Browser Support

Before we can dive more into Grid Layout, it's important to take a look at the status of browser support (http://caniuse.com/#feat=css-grid).



Support in modern browsers

As you can see from the image above, the great news is that CSS Grid is supported by the latest version of most browsers.

If you click the *Show all* option on Can I use, you'll see how far back this support goes. Because Grid is such a new feature, backwards compatibility doesn't extend far, so you need to bear this in mind if there's a chance your audience is using an older version of a major browser.

However, hovering over each version number on Can I use shows the rate of global usage for that version. At the time of writing, the latest version for each browser not to support Grid has a usage of *far less than one percent*.

I ıet Explorer … and other non-supporting browsers

The first proposal of Grid Layout was developed by Microsoft, and IE10 shipped with an `-ms` prefixed implementation. If you take a look at support on Can I use (http://caniuse.com/#feat=css-grid), you'll see that both IE10 and IE11 support CSS Grid … with the caveat that they only support the older version of the specification. Bummer. That's not going to change. But what will change is the usage for those browsers. In July 2018, IE11 usage is down to around 2% globally, and IE10 usage is about a tenth of a percent.

Can I use also indicates that a few other browsers don't support grid layout. These include Opera Mini and Blackberry Browser. Opera Mini is the only one with significant usage, but because it's a mobile browser, you'll most likely want to target it with a mobile-friendly, non-Grid layout anyway.

Fallbacks for non-supporting browsers

Given that the vast majority of browsers in use today support CSS Grid, it's a shame not to use it on the basis that some browsers may not support it. There are various options for using Grid and providing perfectly adequate fallbacks for non-supporting browsers.

Perhaps the simplest option is to provide a simple "mobile first" layout. This will be perfectly adequate for mobile devices that don't support Grid and also desktop browsers like IE.

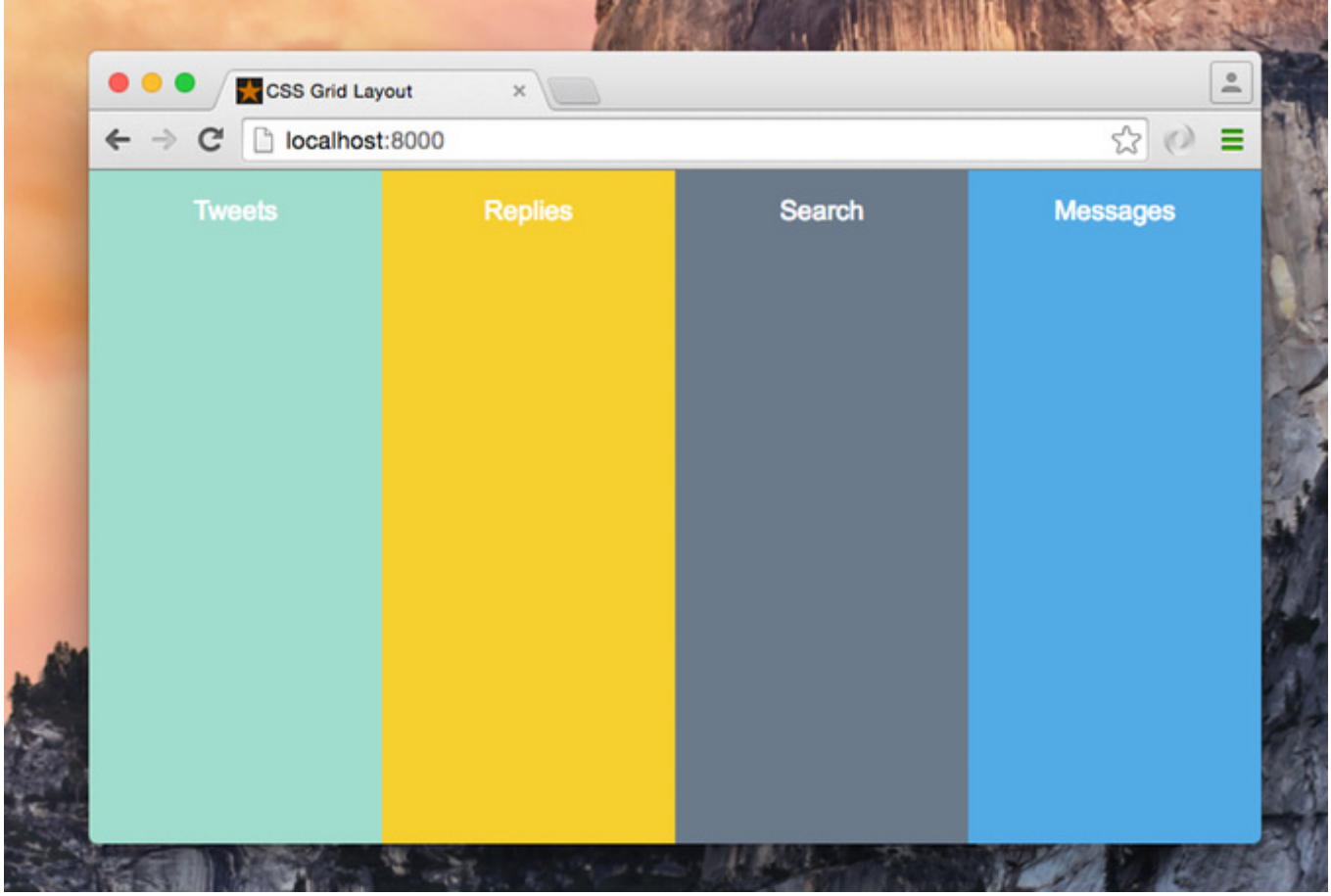
Another option is to use more sophisticated fallbacks and overrides to produce similar results to Grid, as outlined by Rachel Andrew (https://rachelandrew.co.uk/css/cheatsheets/grid-fallbacks).

And a polyfill (https://github.com/FremyCompany/css-grid-polyfill) is also available to provide a working implementation of the Grid Module for browsers that don't support it natively. Note that this might be a bit out of date now.

A Grid Layout Example

Let's start with an example to see the power of Grid Layout, and then I'll explain some new concepts in more detail.

Imagine you want to create a Twitter app with a four, full-height column layout (Tweets, Replies, Search, and Messages), something abstracted and similar to the screenshot below.



(http://www.sitepoint.com/wp-content/uploads/2016/03/1456867693gridlayout04.jpg)

Here is our HTML:

```
<div class="app-layout">
  <div class="tweets">Tweets</div>
  <div class="replies">Replies</div>
  <div class="search">Search</div>
  <div class="messages">Messages</div>
</div>
```

Then we will apply some CSS to the `.app-layout` container element:

```
.app-layout {
  display: grid; /* 1 */
  grid-template-columns: 1fr 1fr 1fr 1fr; /* 2 */
  grid-template-rows: 100vh; /* 3 */
}
```

View a demo here (http://codepen.io/SitePoint/pen/ONPGqz)

Here is the explanation of what we've done in the previous CSS:

- t the display property to `grid`.

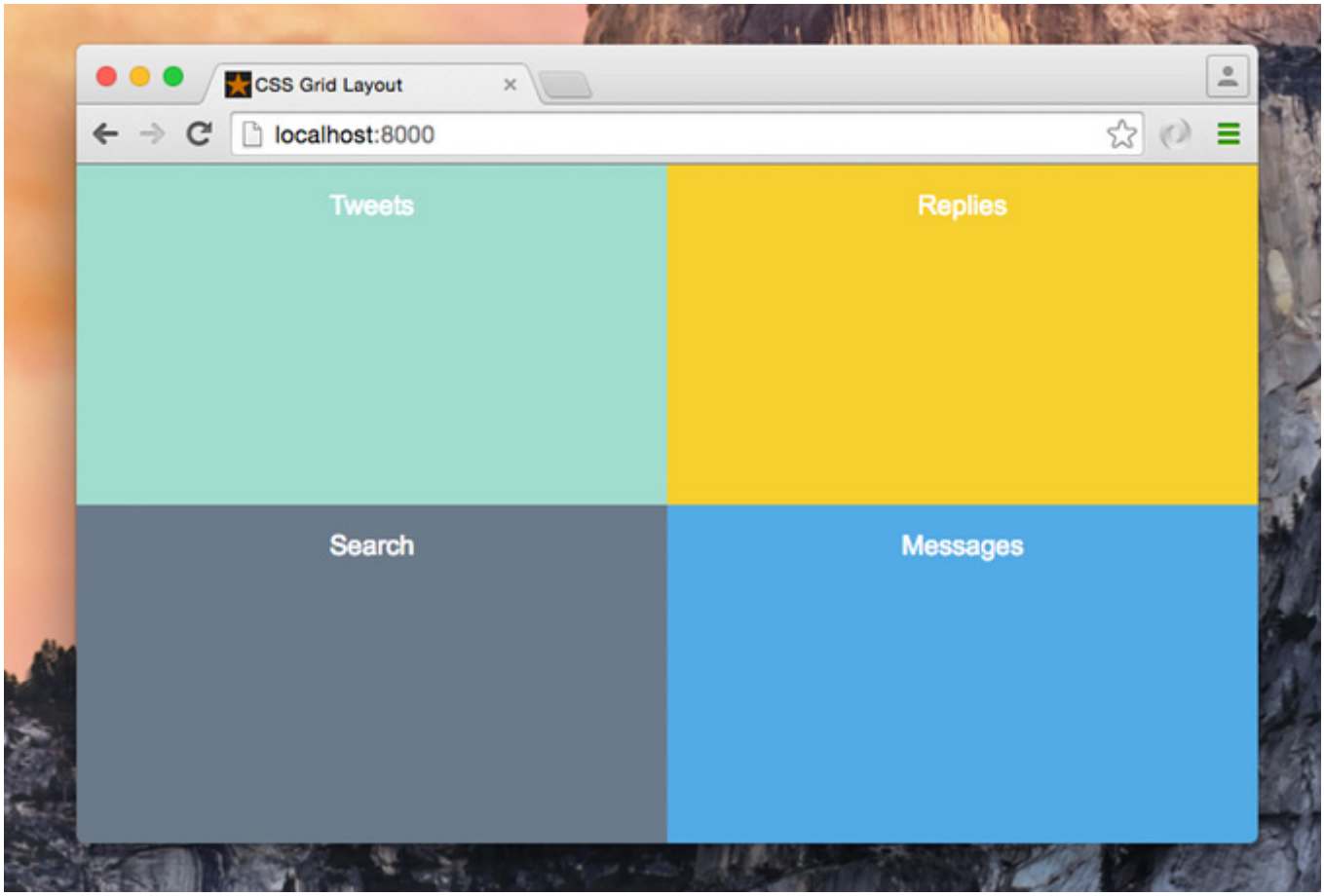
2. Divide the container element into four columns, each column is `1fr` (one fraction) (https://drafts.csswg.org/css-grid/#fr-unit) of the free space within the grid container.
3. Create one row and set the height to be `100vh` (full viewport height).

As you can see, the Grid Layout Module adds a new value to the `display` property which is `grid`. The `grid` value is responsible for setting the `.app-layout` element to be a grid container, which also establishes a new grid formatting context (https://drafts.csswg.org/css-grid/#grid-formatting-context) for its contents. This property is required to start using Grid Layout.

The `grid-template-columns` property specifies the width of each grid column within the Grid, and in our case it divides the `.app-layout` container to four columns; each one is `1fr` (25%) of the available space.

The `grid-template-rows` specifies the height of each grid row, and in our example we only created one row at `100vh`.

A layout with two columns and two rows would look like this:



(http://www.sitepoint.com/wp-content/uploads/2016/03/1456867697gridlayout05.jpg)

And we would use the following CSS:

```
.app-layout {
  display: grid;
  grid-template-columns: 1fr 1fr;
    id-template-rows: 50vh 50vh;
```

View a demo here (http://codepen.io/SitePoint/pen/grbJav)

We can also achieve the above example only on small screens by wrapping the code inside a media query. This opens up a great opportunity for us to customize the layout differently in different viewports. For example, we can create the previous layout only on viewports under `1024px` as follows:

```
@media screen and (max-width: 1024px) {
  .app-layout {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-rows: 50vh 50vh;
  }
}
```

View a demo here (http://codepen.io/SitePoint/pen/aNzrdd)

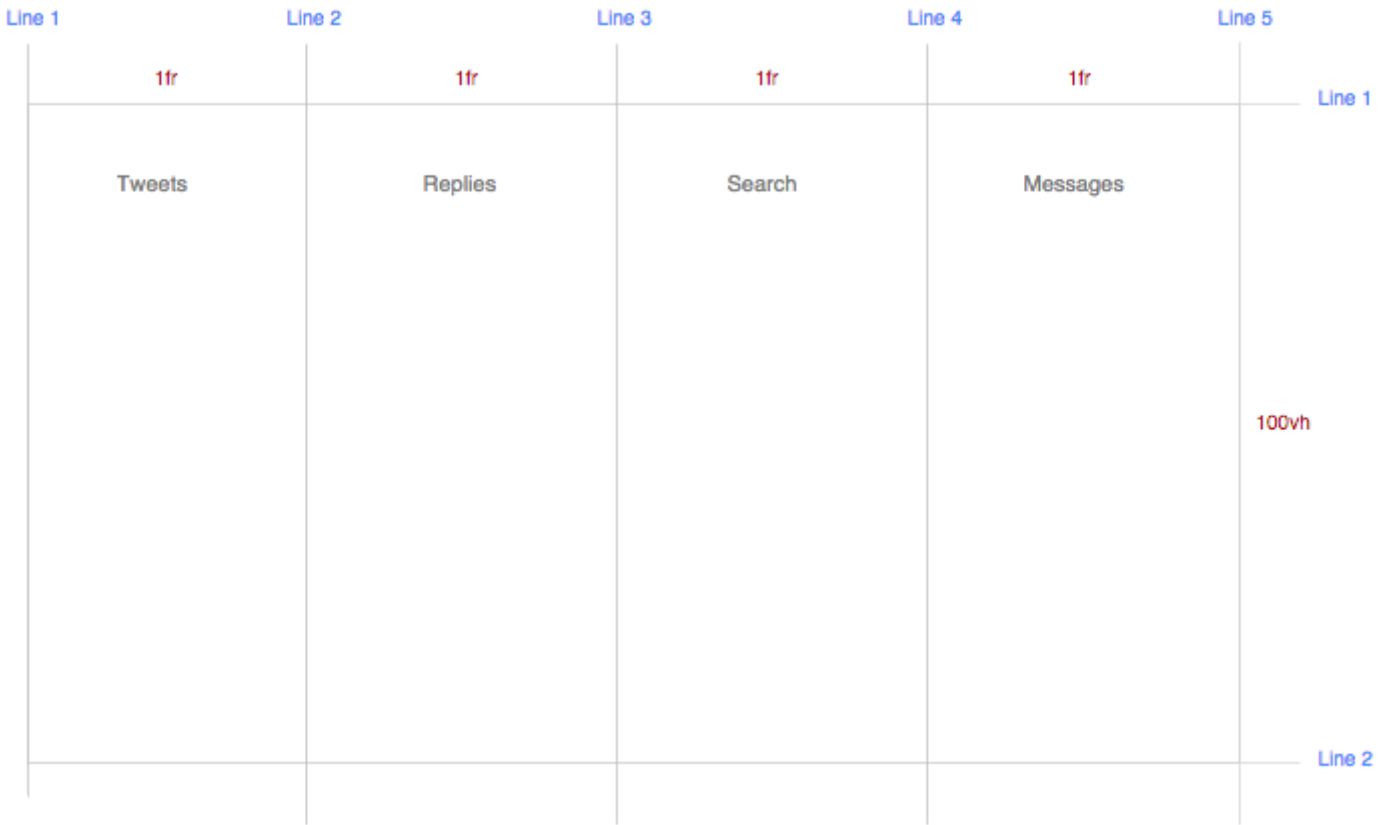
Grid Layout Module Concepts

Now that you've seen a simple example, there are some new concepts that I'd like to cover to give you a better understanding of Grid Layout. Although there are a lot of new concepts, I will only take a look at a few of them.

Grid Item

Grid items are the child elements of the grid container. In the above example, the `.tweets`, and `.replies` elements would qualify as grid items.

Grid Lines



(http://www.sitepoint.com/wp-content/uploads/2016/03/1456867701gridlayout06.png)

A Grid Line is a line that exists on either side of a column or a row. There are two sets of grid lines: One set defining columns (the vertical axis), and another set defining rows (the horizontal axis).

From the above screenshot, which represents the first example, I've created four columns at `1fr` each, which will give us five vertical lines. I also created one row, which gives us two horizontal lines.

Let's see how we can position a grid item inside the grid container.

Position Items by Using a Line Number

You can refer to an exact line number in a grid by using the properties `grid-column-start` and `grid-column-end`. We then give these properties the start and end line numbers.

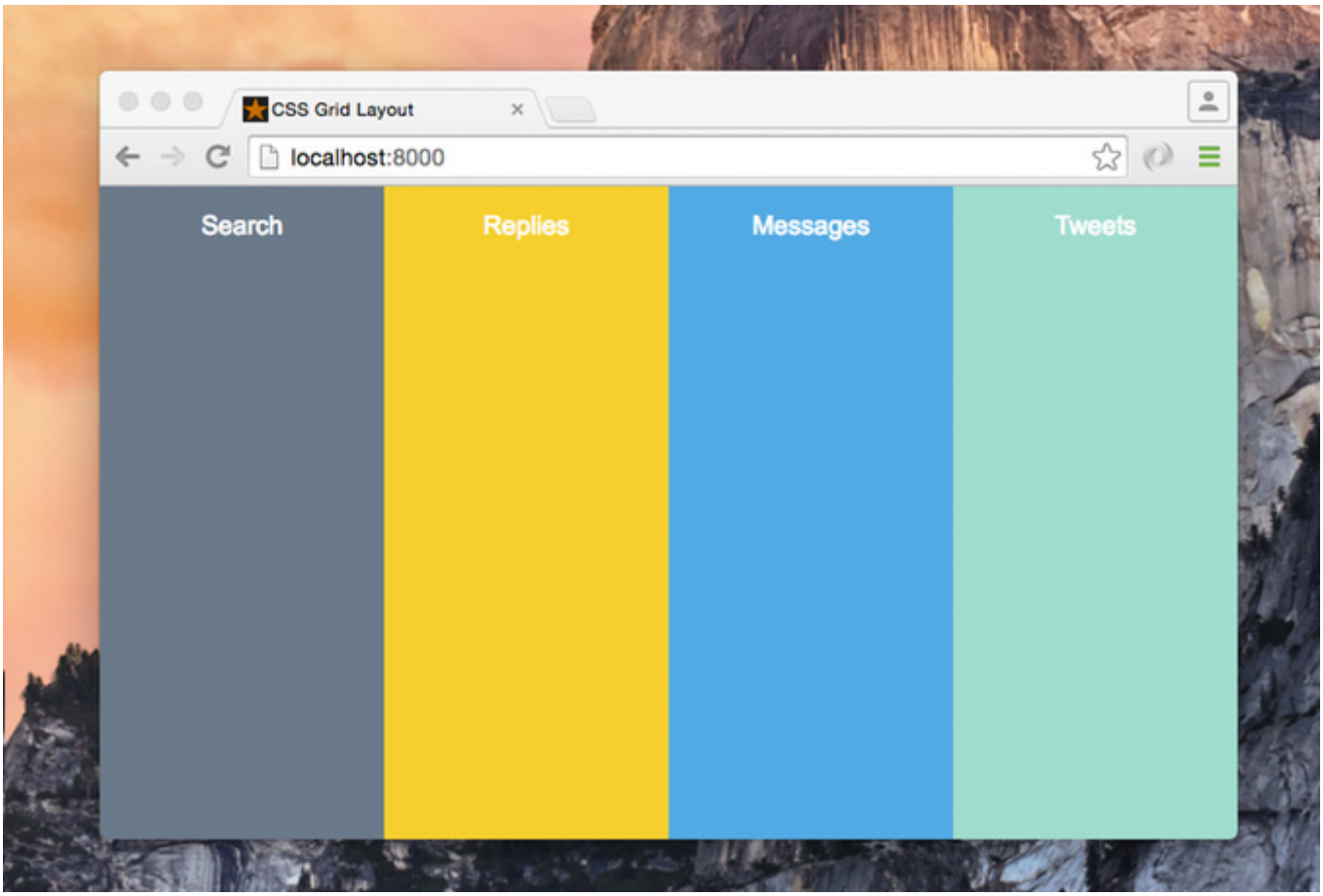
Looking at the previous example, this is how the browser positions the elements by default for us:

```
.tweets   {
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row: 1;
}

.replies  {
  grid-column-start: 2;
  grid-column-end: 3;
  grid-row: 1;
}

.search   {
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row: 1;
}

.messages {
  grid-column-start: 4;
  grid-column-end: 5;
  grid-row: 1;
}
```

Looking at the code for the `.tweet` column, this is what each of the three lines in the CSS does:

1. Position the child element starting from the first vertical line on the left.
2. End the element's position at the second vertical line.
3. Position the element inside the whole row.

You can change this by changing the order of elements with different positions, so the order of the elements will be: `.search`, `.replies`, `.messages`, and `.tweets`.



(http://www.sitepoint.com/wp-content/uploads/2016/03/1456867707gridlayout07.jpg)

And we can do it as follows:

```
.tweets {
  grid-column-start: 4;
  grid-column-end: 5;
  grid-row: 1;
}

.replies {
  grid-column-start: 2;
  grid-column-end: 3;
  grid-row: 1;
}

.search {
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row: 1;
}

.messages {
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row: 1;
}
```

We can also use the `grid-column` shorthand property to set the start and end lines in one line:

```
.tweets {
  grid-column: 4 / 5;
  grid-row: 1;
}

.replies {
  grid-column: 2 / 3;
  grid-row: 1;
}

.search {
  grid-column: 1 / 2;
  grid-row: 1;
}

.messages {
  grid-column: 3 / 4;
  grid-row: 1;
}
```

View a demo here (http://codepen.io/SitePoint/pen/BKyeKV)

This has changed the layout structure with only CSS while the markup is still as it was without any changes. This is a huge advantage of using the Grid Layout Module. We can rearrange the layout of elements independent of their source order, so we can achieve any desired layout for different screen sizes and orientations.

Position Items by Using Named Areas

A grid area is the logical space used to lay out one or more grid items. We can name a grid area explicitly using the `grid-template-areas` property, then we can place a grid item into a specific area using the `grid-area` property.

To make this concept more clear, let's redo the four-column example with the `search` column placed first:

```
.app-layout {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 100vh;
  grid-template-areas: "search replies messages tweets";
}
```

In the last line, we divide the grid container into four named grid areas, each name for a column. The next step is to position each grid item into a named area:

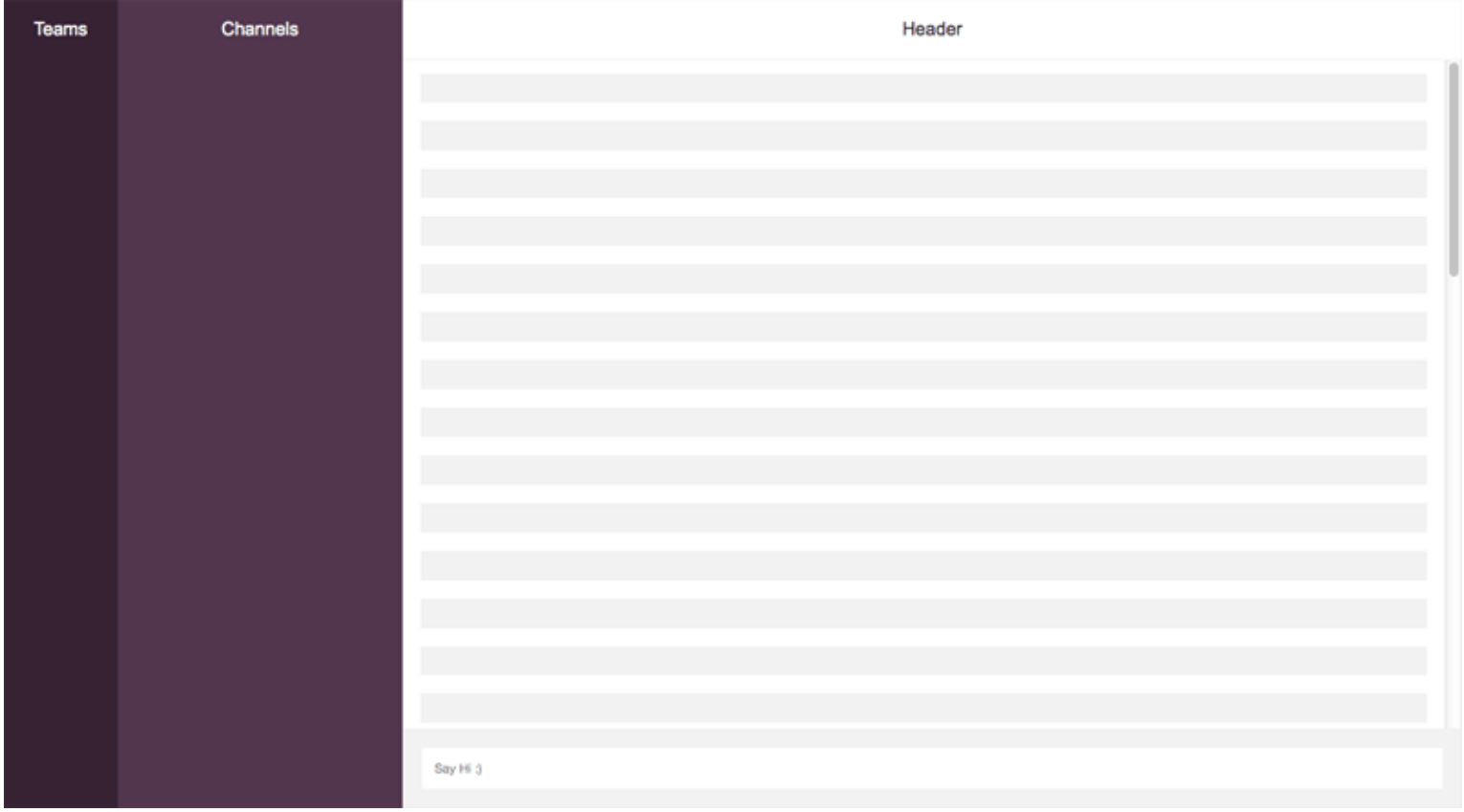
```
.search {
  grid-area: search;
}

.replies {
  grid-area: replies;
}

.messages {
  grid-area: messages;
}

.tweets {
  grid-area: tweets;
}
```

View a demo here (http://codepen.io/SitePoint/pen/LNEoZx)

flojo/poco movimiento

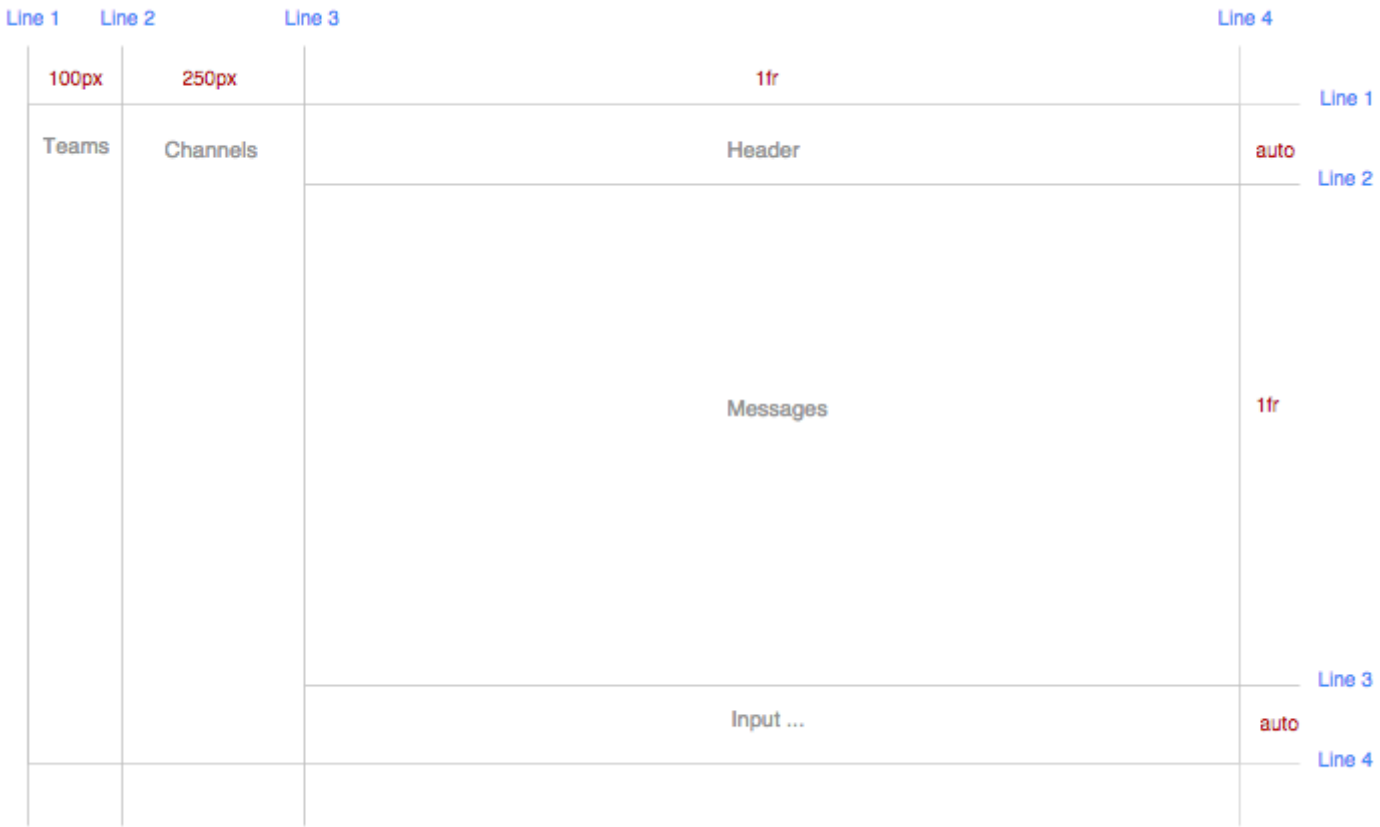
Slack Example

What about using the Grid Layout Module to implement a more complex example, for example, creating the building blocks of the Slack (https://slack.com/) layout. Since we are talking about layouts, we will abstract and simplify the Slack design to the building blocks represented in the grid. Something like this:



(http://www.sitepoint.com/wp-content/uploads/2016/03/1456867712gridlayout08.png)

From this layout we will create three vertical columns, and three horizontal rows, and we can visualize it using the grid lines as follows:



(/ww.sitepoint.com/wp-content/uploads/2016/03/1456867718gridlayout09.png)

Here is the HTML:

```
<div class="app-layout">
  <div class="teams">Teams</div>
  <div class="channels">Channels</div>
  <div class="header">Header</div>
  <div class="messages">
    <ul class="message-list">
      <li></li>
      <li></li>
    </ul>
  </div>
  <div class="input">
    <input type="text" placeholder="CSS Grid Layout Module">
  </div>
</div>
```

And the CSS:

```
.app-layout {
  display: grid;
  height: 100vh;
  grid-template-columns: 100px 250px 1fr;
  grid-template-rows: auto 1fr auto;
}
```

Here I'm using the `grid-template-columns` property to create three columns at 100px, 250px, and the third column takes up the remaining available space. The last line creates three rows: The first and third rows with auto height while the middle row takes up the remaining available space.

The remainder of the CSS looks like this:

```
.teams {
  grid-column: 1;
  grid-row: 1 / 4;
}

.channels {
  grid-column: 2;
  grid-row: 1 / 4;
}

.header {
  grid-column: 3;
  grid-row: 1;
}

.messages {
  grid-column: 3;
  grid-row: 2;
}

.input {
  grid-column: 3;
  grid-row: 3;
}
```

View a demo here (http://codepen.io/SitePoint/pen/MyYdeN)

We can also create the Slack layout using named areas, which you can see in this demo (http://codepen.io/SitePoint/pen/BKyeLP).

Grid Layout Module vs Flexbox

Since many of you have started using Flexbox, you might wonder: When would it be appropriate to use Flexbox and when would it be more appropriate to use Grid Layout?

I found a good explanation from Tab Atkins (https://lists.w3.org/Archives/Public/www-style/2013May/0114.html):

> Flexbox is appropriate for many layouts, and a lot of "page component" elements, as most of them are fundamentally linear. Grid is appropriate for overall page layout, and for complicated page components which aren't linear in their design. The two can be composed arbitrarily, so once they're both widely supported, I believe most pages will be composed of an outer grid for the overall layout, a mix of nested flexboxes and grid for the components of the page, and finally block/inline/table layout at the "leaves" of the page, where the text and content live

Also, Rachel Andrew says (http://www.slideshare.net/rachelandrew/flexbox-and-grid-layout/89):

> Layout for the main page structure of rows and columns.
> ,box for navigation, UI elements, anything you could linearize.

CSS Grid Layout Module Resources

I have not covered all the Grid Layout concepts and syntax, so I recommend you check out the following resources to go deeper:

> CSS Grid Layout Module spec (https://drafts.csswg.org/css-grid/)
> CSS Grid Layout Examples (https://igalia.github.io/css-grid-layout/index.html)
> Grid by Example (http://gridbyexample.com/)
> The future of layout with CSS: Grid Layouts (https://hacks.mozilla.org/2015/09/the-future-of-layout-with-css-grid-layouts/)
> Follow Rachel Andrew (https://rachelandrew.co.uk/) for updates and resources. She is doing a lot of great work in relation to Grid Layout.

Conclusion

As you've seen, the CSS Grid Layout Module is powerful because of its code brevity and the fact that you have the power to change the layout order without touching the markup. These features have helped us to permanently change the way we create layouts for the web.


(htt
s:

Meet the author
Ahmad Ajmi (https://www.sitepoint.com/author/aajmi/)
(https://twitter.com/AhmadAjmi) (https://www.facebook.com/ahmadajmii)
(http://www.linedin.com/in/ahmadajmi) (https://github.com/ahmadajmi)

Ahmad Ajmi (http://ahmadajmi.com/) is a self-taught front-end developer passionate about the Web, open source, and programming.

- FAQ (https://sitepoint.zendesk.com/hc/en-us)
- Publish your book with us (https://sitepoint.typeform.com/to/HtAXVN)
- Write an article for us (https://sitepoint.typeform.com/to/DMmYfn)
- Advertise (/advertise/)
- Privacy policy (/privacy-policy/)

Connect

 (https://www.facebook.com/sitepoint)  (http://twitter.com/sitepointdotcom) 

(https://www.sitepoint.com/feed/)  (https://plus.google.com/+sitepoint)

Privacy