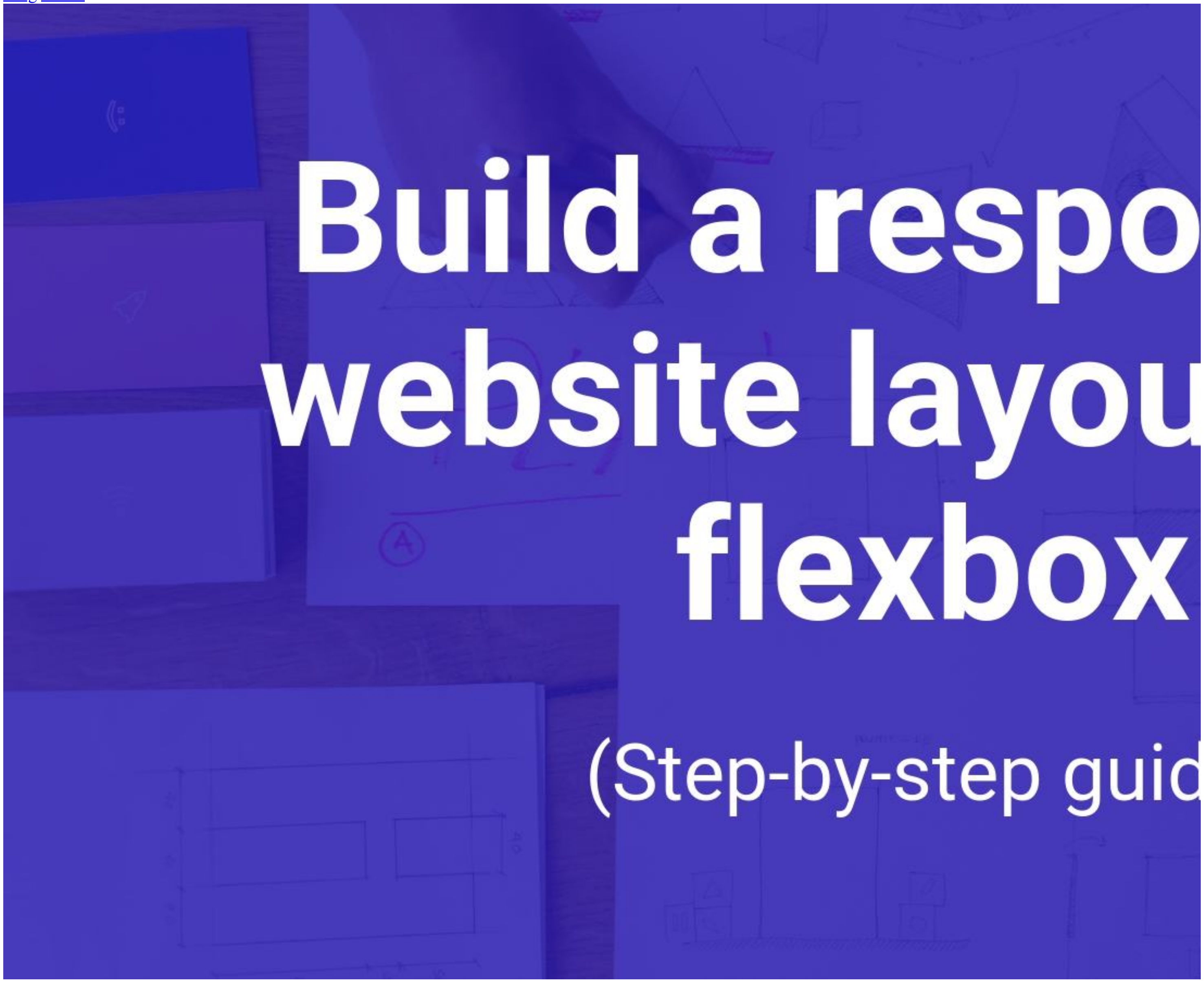


Build a responsive website layout with flexbox (Step-by-step guide)

Published: September 21, 2018
Updated: October 1, 2019

Flexbox is a relatively new front-end feature that makes building a website layout (and making it responsive!) much, much easier than it used to be.

In days past, to build a website, you would have to use float grids or even tables to make your layout look like it should. And those methods aren't the best for responsive design– making sure the website looks good across desktop, tablet, and mobile devices.

If you want to stay current with web development trends, you definitely want to know how to use flexbox.

Because float grids are quickly becoming a thing of the past.

This step-by-step guide will take you through the process of building a simple responsive website layout.

Here's a quick look at what you can expect to go through in this tutorial:

Steps to build a simple website layout

1. Sketch out how the layout will look on mobile, tablet, and desktop.
2. Start coding the basic layout, using semantic HTML and CSS.
3. Going section by section, build the rest of the layout.
4. In your CSS, follow a mobile-first approach, creating the styles for the smallest widths, then progressively greater widths.

I'll explain my thought processes as I go, and share what I've found to be good practices.

Sound good? Let's get started!

Wireframing out the website layout

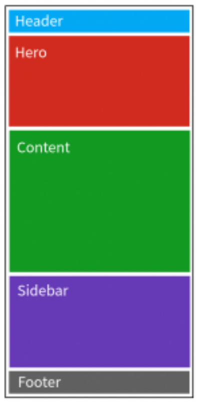
Wireframes are diagrams of all the organizational parts of your website. They can be super detailed, almost like designs, or they can be basic and just document the main aspects.

For our purposes here, we're going to have a very basic wireframe. We're going to break up the website into its core sections and decide how each section will look on mobile, tablet, and desktop.

The sections we will be building contain the Header, Hero, Content section, Sidebar, and Footer.

I use an online tool called MockFlow to create my basic wireframes.

Here's the mobile layout:



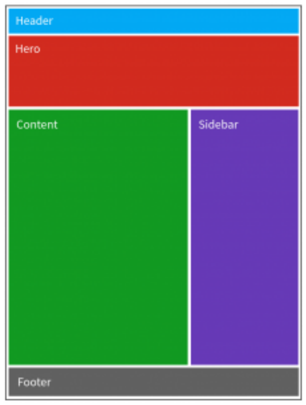
Click for full-size

You can see that all the sections basically stack on top of one another in one long column, including the sidebar.

Stacking is the most basic way to fit content efficiently onto a narrow device like a mobile phone.

It wouldn't make much sense to try to make the sidebar next to the regular content, because phones just aren't wide enough to fit both side-by-side.

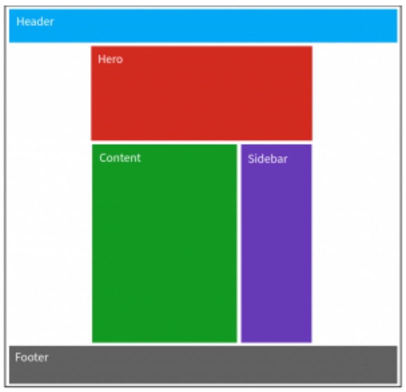
Moving up in width now– here's the tablet layout:



Click for full-size

The main difference here is that because tablets are much wider than phones, we can now fit the sidebar next to the main content section. And all sections take up the full width of the tablet.

And for the widest device– here's the desktop layout:



Click for full-size

For desktop, you have to start to consider how your website will look on very wide monitors. Especially nowadays with ultrawide screens gaining in popularity.

If we had the website content extend full-width on a large monitor, it would make it harder to read and scan the content.

Imagine forcing your eyes to travel allll the way from the left to the right side. It's way too much work, and it would drive away your users.

In order to make the website still readable on wide screens, we've capped off the content at a certain width, and centered it. This will make it much easier and intuitive to read no matter how big the monitor is.

> **Give your website visitors a good user experience:**
>
> As website builders, we want visitors to stay on our website and not leave it. To accomplish this, we have to make our sites be easy to navigate. In marketing and web development lingo, we call this having a good user experience or UX for short.
>
> Basically, try to put yourself in the shoes of your visitors. You want their experience to be efficient and hopefully enjoyable! Don't make them have to look too hard for how to navigate or read the website.

Now that we know what we want the website to look like, we'll start the fun part– building everything out in HTML and CSS!

Building the basic structures and styles

Map out the layout with HTML elements

Working from the wireframes we've made, we will create an HTML element for each section that was in the wireframe.

Here's the markup in the `<body>` that we're starting with:

```
<!-- Main Content -->
<main>
    Main

    <!-- Header -->
    <header>
        Header
    </header>

    <!-- Hero -->
    <section>
        Hero
    </section>

    <!-- Content -->
    <section>
        Content
    </section>

    <!-- Sidebar -->
    <aside>
        Sidebar
    </aside>

    <!-- Footer -->
    <footer>
        Footer
    </footer>

</main>
```

You can see how each element from the wireframe now has a corresponding element in the HTML. And I've just put in simple placeholder text, no real content.

> **Start out simple**
>
> Even when you're building a real website, not just a simple layout like we are here, it's best to start out with the basic skeleton. Then you can gradually go back over and fill out the details.
>
> This approach is more efficient in the long-term. Starting with a sketch makes it easier to change directions before you've committed too much.

Start adding basic CSS properties

Now let's dump in some super basic CSS so we can start making this layout look good!

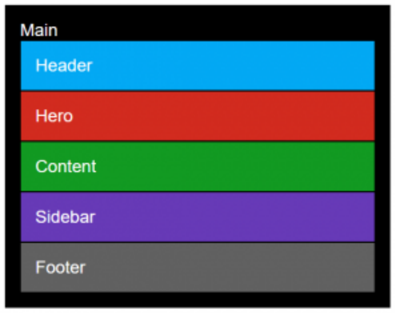
```
main, header, section, aside, footer {
    margin: 0;
    padding: rem-calc(20);
    border: 1px solid #000000;
    color: #ffffff;
}

main {
    background: #000000;
}

.header {
    background: #03a9f4;
}

.hero {
    background: #d22b1f;
}
```

```
.content {
    background: #129a22;
}

.sidebar {
    border: 1px solid #000000;
    background: #673ab7;
}

.footer {
    border: 1px solid #000000;
    background: #616161;
}
```

Not shown are some very general styles, like setting box-sizing: border-box and font styles. But for the purposes of this article, these are the only styles you need to worry about.

I like to add borders to my elements to make it easier to identify where each element is and troubleshoot any weird issues. And I've also added background colors to match the wireframe mockups we did earlier.

If we open the HTML file in the browser, here's what we'll see!



You might have noticed this already, but the website looks pretty much identical to the mobile wireframe. Everything by default is stacked on top of one another.

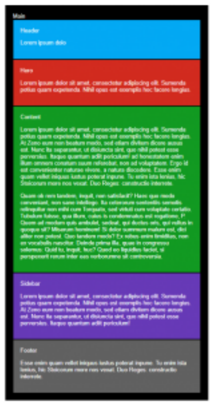
Add in some placeholder content

After you've created your HTML elements, it's also a good idea to add some placeholder content. This will make the website look more similar to what it would like when you're finished.

You don't have to get too fancy– we're just going to copy and paste some lorem ipsum filler text into each element. For example, here's what's in the Hero element:

```
<section class="hero">
    Hero
    <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sumenda potius quam expetenda. Nihil opus est exemplis hoc facere longius.
    </p>
</section>
```

After adding the lorem ipsum to all the elements, here's what it will look like in the website:


Click for full-size

Looking good!

You'll notice that I added different lengths of placeholder text to each element, to mirror what the final content would look like.

Optimizing the CSS for mobile

All right, we're basically done with building the layout for mobile!

One more tip for making a website look good on mobile is to add a uniform padding around the sides (and the top and bottom, if you desire).

This just gives a tad bit of breathing room for the user. If there was zero padding and zero margin, then the content would be right up against the edge of the screen, which would feel cramped.

You don’t want to add too much space, in order to keep the content readable. In this case I have a global padding of 20px, but you can add a padding of 10px, 15px, or whatever you think looks best.

Let’s move on to adding styles for tablet view.

Creating a two-column layout for tablet

If we refer back to the tablet wireframe, we’ll see that it has the Content and Sidebar elements side by side. Everything else is stacked vertically just like on mobile.

We need to add some styles so the Content and Sidebar are formatted into two columns. We’ll be using flexbox for this, as opposed to CSS grid.

> **Flexbox and CSS grid both have strengths and weaknesses**, and can be very powerful when used together. Generally, flexbox is good at aligning elements in a single row or column, or multiple rows where you want everything spaced evenly. CSS grid is great at complex layouts involving both rows and columns, and if you want to make your content stick to a specific grid layout. There’s a great article on MDN that talks about when to use flexbox or CSS grid.

First of all, we are going to change our HTML and wrap the Content and Sidebar elements in a parent `<div>` that we’ll give a class of “flex-container.” (Using ellipses to stand for additional markup that’s not really important right now)

```
<div class="flex-container">

   <!-- Content -->
   <section class="content">
      Content
      ...
   </section>

   <!-- Sidebar -->
   <aside class="sidebar">
      Sidebar
      ...
   </aside>

</div>
```

Before we write our CSS, we need to decide just how we want the Content and Sidebar to behave when we’re in the two-column layout. There are a few different options, and again the “right” answer is dependent on your own situation.

What I want is for the Sidebar to always be 300px wide, and the Content to take up the rest of that space.

To do this, we’ll add our flexbox styles in the CSS (again using ellipses to stand in place of the additional code):

```
@media screen and (min-width: 640px){
    .flex-container {
        display: flex;
    }
}
```

Using a media query, we will turn on flexbox when the device width is 640px and above. Meaning that on smaller widths on phones, it will remain stacked. Once it hits that 640 mark it will switch to the flexbox layout.

```
.content {
    flex: 1;
    ...
}

.sidebar {
    flex: 0 1 300px;
    ...
}
```

For the Content and Sidebar elements, we’ve added the flex property.

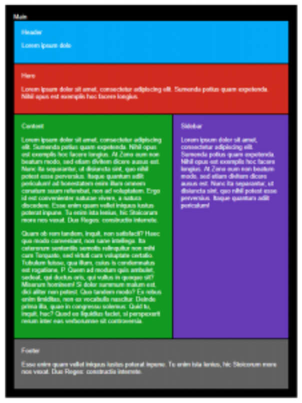
We want the Content width to be calculated by flexbox, so we’ll set it to `flex: 1`, which is shorthand for `flex-grow: 1`, `flex-shrink: 1`, and `flex-basis: 0%`.

This is the “default” flexbox setting– if you set all child elements to have `flex: 1`, their widths will be calculated and distributed as evenly as possible.

On the Sidebar, to set its width to 300px, we will use `flex: 0 1 300px`. According to CSS-Tricks, that last property, flex-basis, “defines the default size of an element before the remaining space is distributed.”

This will ensure that the Sidebar will always be 300px, and the rest of the space will be distributed to the Content section.

Here’s the result of that!



Click for full-size

Now let’s add our styles for desktop.

Cap the content width for desktop

If we look back at the tablet and desktop wireframes, they look pretty similar. Both have the content next to the sidebar.

The main difference for desktop, as mentioned earlier, is to set a max-width for the main content. This will ensure that the website is readable even on ultrawide monitors.

Helper classes in CSS

Since we will likely need this set of styles for multiple elements, let’s create a helper class which will be easily reusable.

We’ll add our new class and accompanying styles in our CSS like this:

```
.wrapper {
    margin: auto;
    max-width: rem-calc(1200);
}
```

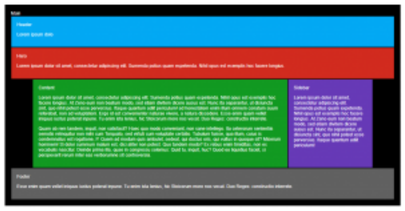
This will set the max-width of the element to 1200px (converted into rem units), and also center it on widths larger than 1200px.

In the HTML we will add the wrapper class to the flex-container wrapper, since we want to limit the width for the Content and Sidebar container.

The markup will then look like this:

```
<div class="flex-container wrapper">
    …
</div>
```

And if we check out the website on a larger width, we can see that there is extra space around the Content/Sidebar wrapper, just like we wanted 😁



Click for full-size

In conclusion

So there you have it– we’ve made a simple responsive layout that looks good on mobile, tablet, and desktop!

If you’d like to see the website in action, you can check out a demo here.

And you can download all the code for this project from my GitHub here.

*Note: the project uses Gulp and Sass. If you need help installing Gulp, check out my Simple Gulp Tutorial here.

Thanks for reading! Let me know what you think of this tutorial in the comments below 😆

Want to learn how to build a website?

I'm making a course that will teach you how to build a real-world responsive website from scratch!

Learn more

Related posts

Super simple Gulp tutorial How to layout and design a website (without any design skills!) Learn web development as an absolute beginner (2020)

Comments

R
Ryu
Mar 31, 2019 at 4:16 pm

Hello there.
Overall it's a nice guide but there is one thing I would like to mention.
You haven't written that you use some CSS pre-proc, as rem-calc(20) wouldn't work at all for me. :)

Reply
Cancel Reply
Email
Name
Comment
Submit
J
Jessica Chan
Apr 4, 2019 at 9:24 am

Hi Ryu, you're right-- if you clone the GitHub repo I included a mixin to convert the rem-calc() to pixels, but I should have mentioned that. Thanks for the feedback!

Reply
Cancel Reply
Email
Name
Comment
Submit
P
Peter Scott
Apr 8, 2019 at 12:57 pm

Dear Ryu,

I like your tutorial!

I would like to put several row items in the "content" column, each separated from the others by, say, a 20px space. How would I do that? (I don't use gulp or SASS, just basic Skeleton.)

Thanks for any hints.

Reply
Cancel Reply
Email
Name
Comment
Submit
J
Jessica Chan
Apr 9, 2019 at 4:38 pm

Hi Peter-- you should be able to add child elements in the "content" element, for example:

```
<div class="content">
<div class="row">
... content ...
</div>
<div class="row">
... content ...
</div>
```

```
<div class="row">
... content ...
</div>
</div>
```

Reply
Cancel Reply
Email
Name
Comment
Submit
P
Peter Scott
Apr 8, 2019 at 1:01 pm

Oh, I forgot to say "Dear Ryu and Jessica"! My bad.

Reply
Cancel Reply
Email
Name
Comment
Submit
P
Peter Scott
Apr 11, 2019 at 4:19 pm

Dear Jessica,

Oh this is easier than I thought. Thanks!

Reply
Cancel Reply
Email
Name
Comment
Submit
M
Mohamed
Aug 14, 2019 at 1:16 pm

You didn't specify any height to these header ,footer,section and aside how did it took the desired height

Reply
Cancel Reply
Email
Name
Comment
Submit
J
Jessica Chan
Aug 20, 2019 at 9:32 am

Hi Mohamed, I added a static height to the elements just for illustration purposes. But normally each one would have content that would determine the height.

Reply
Cancel Reply
Email
Name
Comment
Submit
D
Dave
Aug 17, 2019 at 10:22 am

Hi Jessica,

Thank you for this well-written walk-thru. It's the clearest I've read on flexbox and I think I get it now.

Isn't the media query inverted, though?

Reply
Cancel Reply
Email
Name
Comment
Submit
J
Jessica Chan
Aug 20, 2019 at 9:31 am

Hi Dave, thank you! You can use either min- or max-width or both in your media queries. I personally like only using min-width (this is what Zurb Foundation does) so that you won't have overlap in the breakpoints. Hope this helps!

Reply
Cancel Reply
Email
Name
Comment
Submit
T
Thomas Fitzpatrick
Aug 18, 2019 at 9:57 am

Hi Jessica
A tutorial on basics but very helpful indeed. Many thanks.

Reply
Cancel Reply
Email
Name
Comment
Submit
J
Jessica Chan
Aug 20, 2019 at 9:14 am

Thank you!

Reply

Cancel Reply
Email
Name
Comment
Submit
K
Kassey
Aug 20, 2019 at 6:33 am

Thanks Jessica for the simple illustration of use of Flexbox for responsive design. This is the best explanation I have found in my internet search.

You show us how to display "content" and "sidebar" in same row for both tablet and desktop. Can you expand the concept further as shown below?

Tablet:
"hero" and "content" in one row
"sidebar" and "footer" in another row

Desktop:
"hero", "content", "sidebar", "footer" in one row

Thanks.

Reply
Cancel Reply
Email
Name
Comment
Submit
J
Jessica Chan
Aug 20, 2019 at 9:14 am

Hi Kassey, you're very welcome! That layout would be possible in flexbox if you wrap the hero and content in a parent div, and the sidebar and footer in another parent div. However this type of layout might be better suited to CSS grid. Also I'm not sure if putting everything in one row would be recommended, even for desktop (unless you are going for the horizontal scrolling effect). Best of luck!

Reply
Cancel Reply
Email
Name
Comment
Submit
A
Andrew
Feb 15, 2020 at 8:29 pm

This is excellent!
Thanks

Reply
Cancel Reply
Email
Name
Comment
Submit