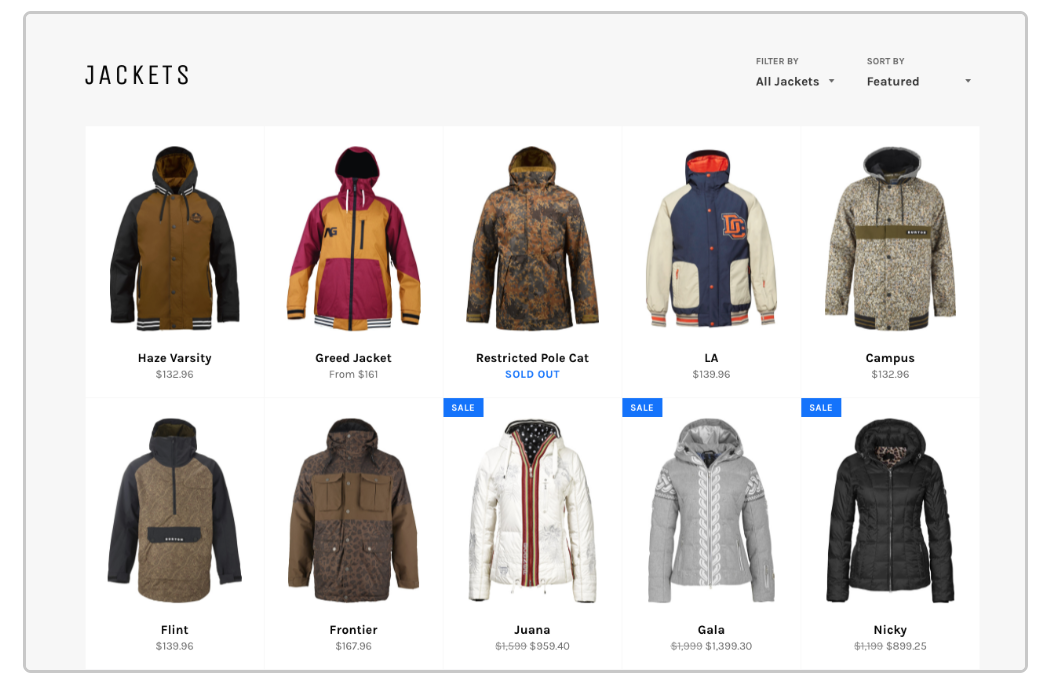
Flex box product Page

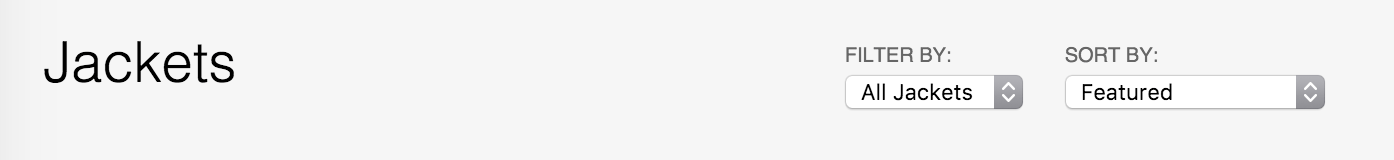
[You are here on Products Layout](#_You_are_Here_1)

<https://css-tricks.com/designing-a-product-page-layout-with-flexbox/>



**The Header Layout**

The first thing we want to do is set up our filter navigation which contains our heading and two filter elements (dropdowns) with labels.



Let’s start with setting up a flexbox container with its content:

# HTML

<nav class="product-filter">

<h1>Jackets</h1>

<div class="sort">

<div class="collection-sort">

<label>Filter by:</label>

<select>

<option value="/">All Jackets</option>

</select>

</div>

<div class="collection-sort">

<label>Sort by:</label>

<select>

<option value="/">Featured</option>

</select>

</div>

</div>

</nav>

Our .product-filter will be our flex container, so we can align the flex item child elements accordingly. We declare the flex container as follows:

# CSS

.product-filter {

display: flex;

}

Our <h1> element is given a flex-grow value of 1 so that it both expands the flex container to full width and expands itself to full the remaining space (which right-aligns the sorting dropdowns).

.product-filter h1 {

flex-grow: 1;

}

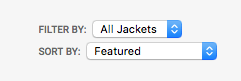
To horizontally align the child elements of our .sort container, we we’ll make it a flex container too. You can nest flex containers!

.sort {

display: flex;

}

The sorting containers, being <div>s, will stack on top of each other by default:



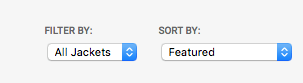
By default, display: flex; will align child elements horizontally. We’ll use that on the sorting container to align the side-by-side. We’ll make each individual sorting container a flex container too (a third nested flex container!) and *also use* flex-direction: column; for the filters so they align vertically:

.collection-sort {

display: flex;

flex-direction: column;

}



In a few lines of CSS our heading and filters are designed the way we want. Now with our current knowledge of flexbox, we’ll work on our grid layout for our products.

# You are Here

**The Products Layout**

We’ll use the following HTML for our Flexbox Layout:

# HTML

<section class="products">

*<!-- It's likely you'll need to link the card somewhere. You could add a button in the info, link the titles, or even wrap the entire card in an <a href="..."> -->*

<div class="product-card">

<div class="product-image">

<img src="assets/img/coat-01.jpeg">

</div>

<div class="product-info">

<h5>Winter Jacket</h5>

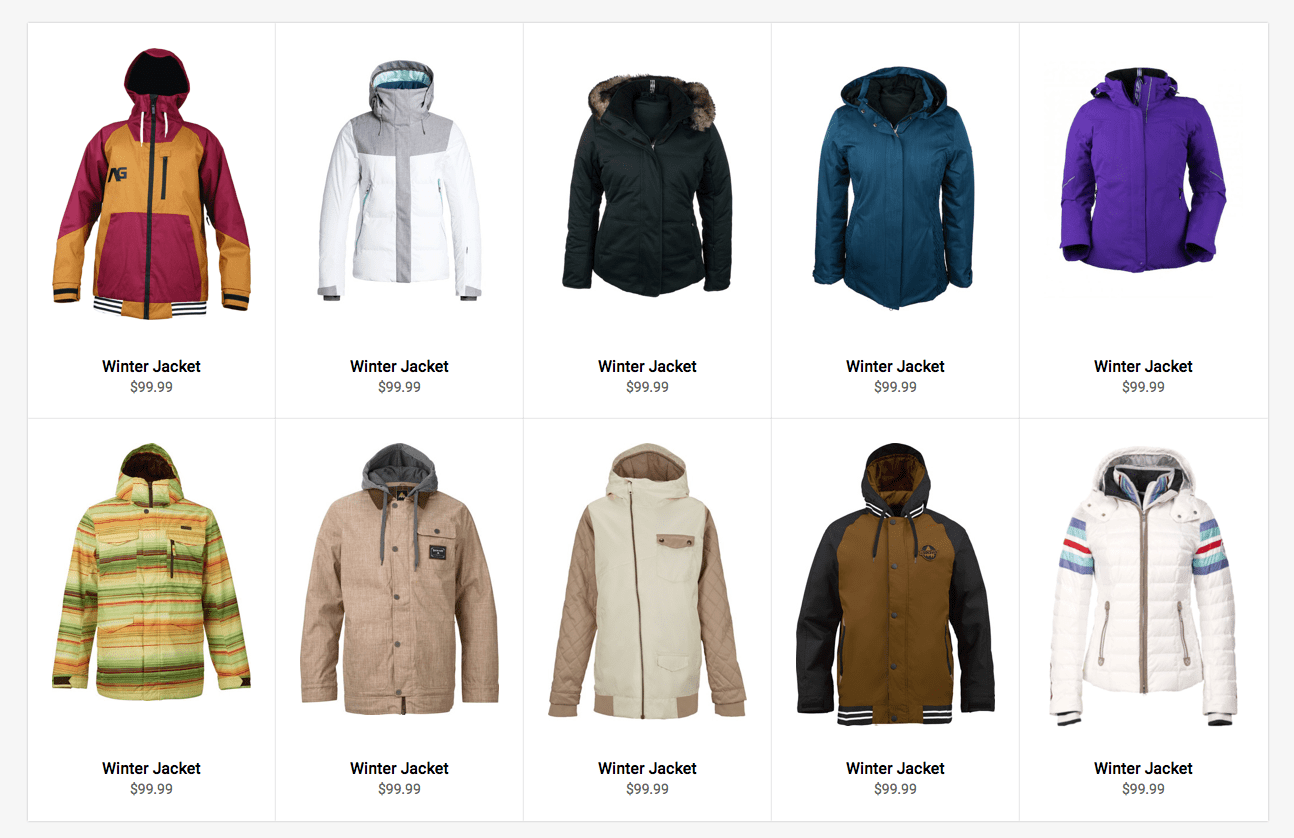
<h6>$99.99</h6>

</div>

</div>

*<!-- more products -->*

</section>



Like before, we need a flex container. In this scenario we will use .products as our flex container. We’re going to add two new properties which will allow the flex children to align horizontally *and wrap* into new rows as the viewport width expands and shrinks:

# CSS

.products {

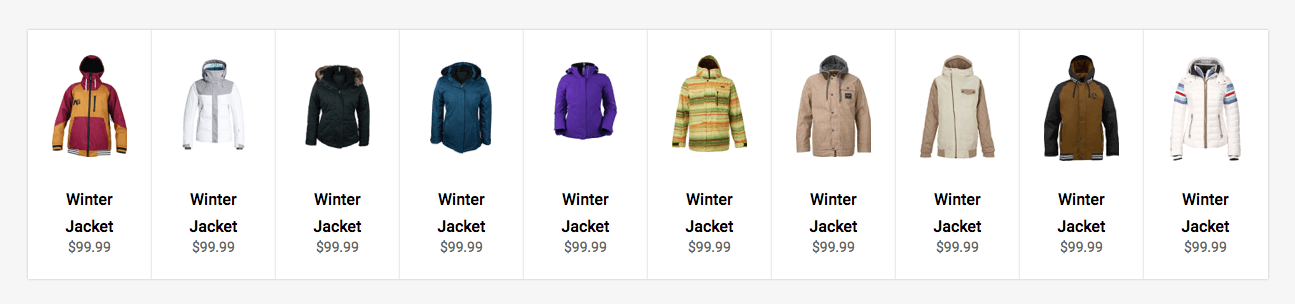
display: flex;

flex-wrap: wrap;

}

By default, display: flex; will lay out children horizontally starting to the left, but we’ve added flex-wrap: wrap; to wrap the children to a new row once there is not enough space to fit the elements on the same row depending on viewport width.

To start exerting control over the width for our flex items, we will add flex: 1; so that all of our flex items take equal space in the row. In our example we have 10 jacket products, and adding flex: 1; will place all products on one row.



For our design, we want 5 items per row and to wrap the rest to new rows as needed. To get five per row, they’ll need to have a width of 20% (5 \* 20 = 100). Settings flex-basis: 20% would do the trick, but when we factor in padding, it exceeds 100% and we’ll only get 4 per row. With 2% padding on either side and 16% flex-basis, it’ll be just right.

# CSS

.products {

display: flex;

flex-wrap: wrap;

}

.product-card {

padding: 2%;

flex-grow: 1;

flex-basis: 16%;

display: flex; */\* so child elements can use flexbox stuff too! \*/*

}

We can also do this in shorthand as follows:

.product-card {

flex: 1 16%;

}

Having flex-grow at 1 for the products will ensure that the row of products always fills the entire space.

To make sure the images within fit nicely:

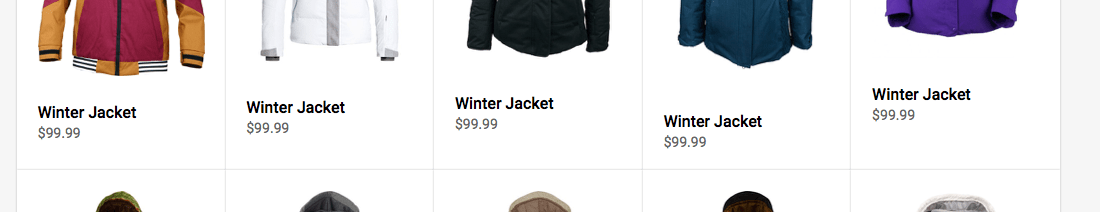
.product-image img {

max-width: 100%;

}

**Bottom-Aligning the Product Footers**

It can sometimes be tricky to set a fixed footer or set content to the bottom of a container. Flexbox can help us there as well. If you have been following along with the code bit by bit, you’ll see that the label for our jackets are not perfectly aligned underneath the jacket images.



A step ladder effect generated by the variable height of the jacket images.

This type of scenario is common: we can’t control the height or length of content, but would like to have another element perfectly set to the bottom of the container. Flexbox is already helping us keep the containers themselves of equal-height-per-row, but the images are still of variable height.

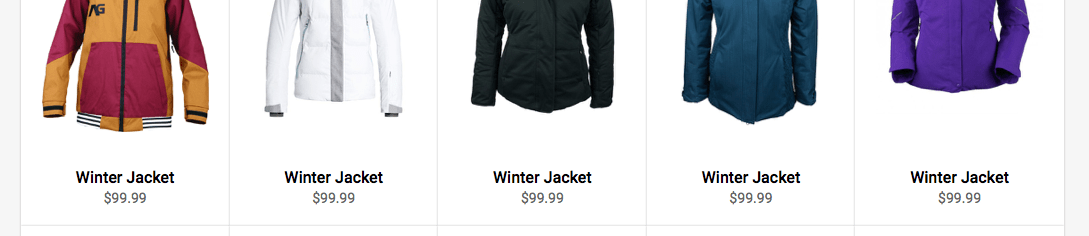
Popular methods to align the bottoms may require using absolute positioning or even JavaScript. Fortunately, with flexbox this complex task can be accomplished by simply adding the following CSS to our .product-info container:

.product-info {

margin-top: auto;

}

That’s it! Flexbox is smart enough to then place the element to the bottom of the Flex container. With a couple of lines of styles we get the following result:

Nicely aligned bottoms.

**Responsive Flexboxing**

As we have less horizontal space to work with, we’d like to reduce the number of products per row. For example, if the max viewport is 920px, we would like the number of items per row to be limited to four which we can accomplish with the following:

# CSS

@media (max-width: 920px) {

.product-card {

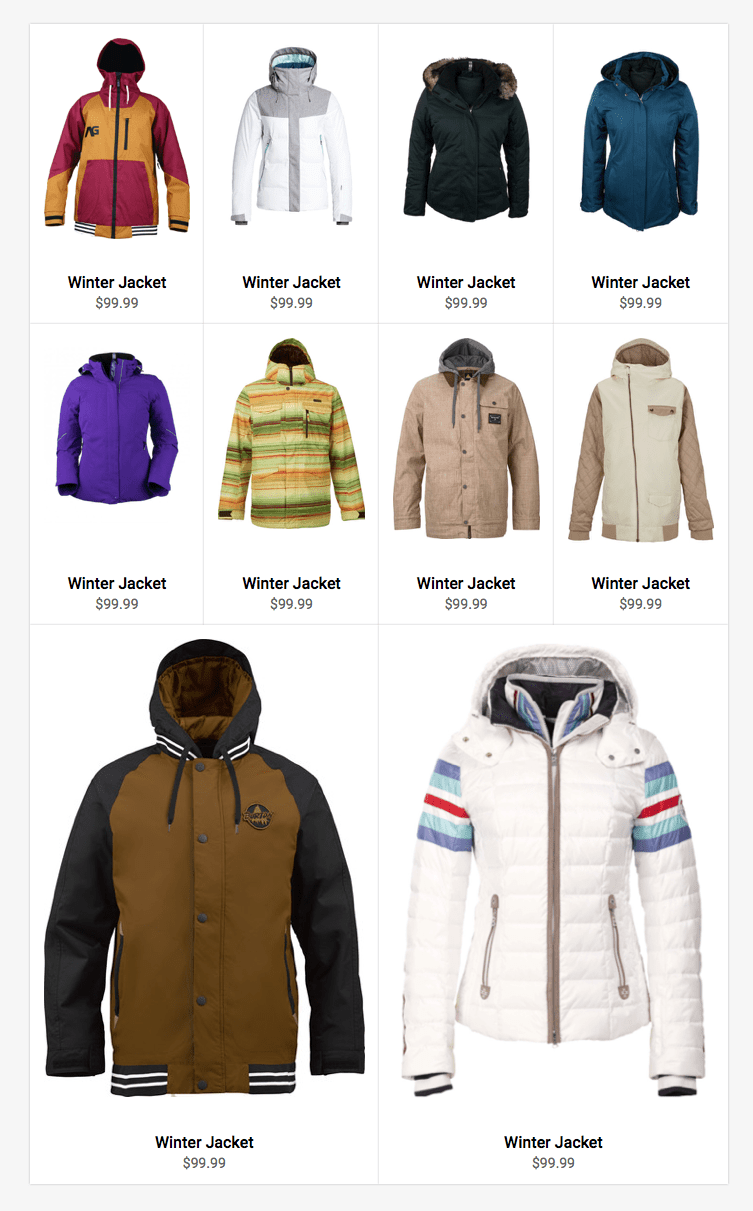
flex: 1 21%;

}

}

(Remember it’s not 25% (100% / 4) because we have to compensate for the padding that I added earlier. We could avoid this with [box-sizing: border-box](https://css-tricks.com/inheriting-box-sizing-probably-slightly-better-best-practice/), but that’s your call.)

The previous lines of CSS almost give us the desired result we want because we get four items per row. But, the *last* row has two large items.



Flexbox is smart enough to fill any available space, something that we don’t have to worry about with other layout methods. To improve the layout for this viewport, we would prefer to have larger images of the jackets at the top versus the bottom to better highlight the products.

One of the ways we can enlarge the first to instead of the last two is to select them and change their size directly:

# CSS

*/\* Select the first two \*/*

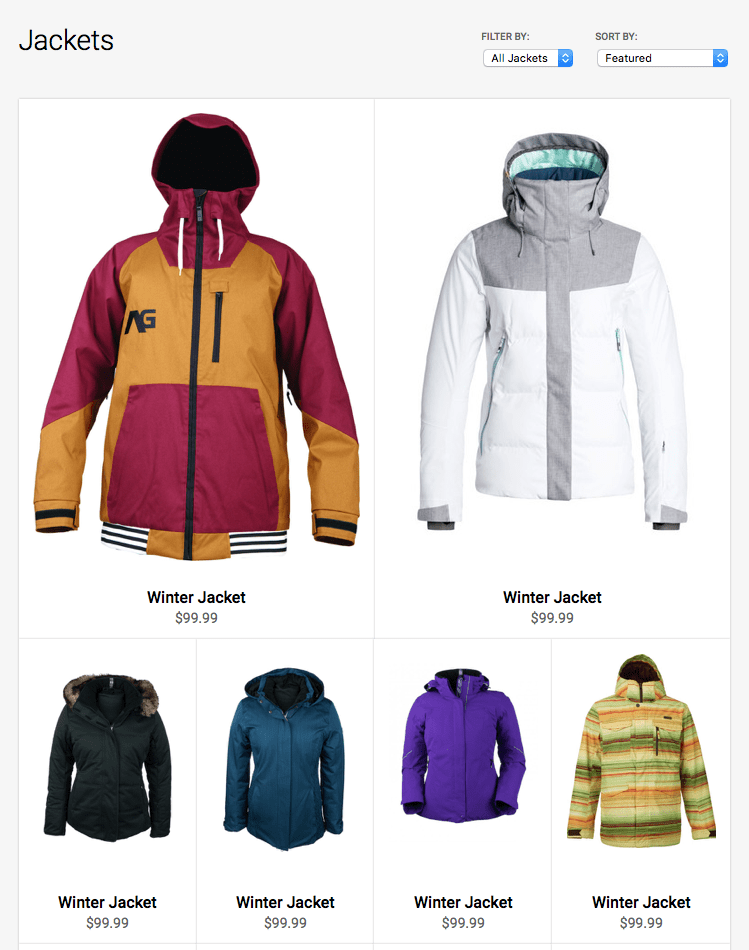
.products .product-card:first-child,

.products .product-card:nth-child(2) {

flex: 2 46%;

}

Now with our CSS applied we get a really great layout that is optimized for smaller viewports like iPads in Portrait mode.



For even smaller viewports we would prefer a two column layout for the jackets. We can accomplish this layout with the following media query:

# CSS MEDIA Queries

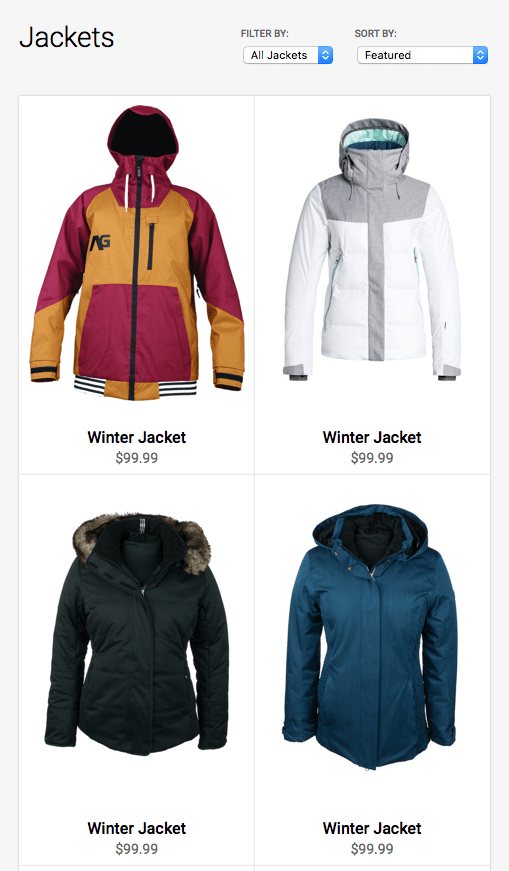
@media (max-width: 600px) {

.product-card {

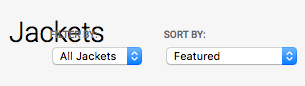
flex: 1 46%;

}

}



If we now view our page on a smaller viewport like an iPhone 6, we will see that our filter nav bar is overlapping our heading.



This is happening because our .product-filter is set to contain all our flex items in a horizontal line, no matter how many items it contains (no wrapping). With the following code, we can easily change this with a media query so that our content is set vertically:

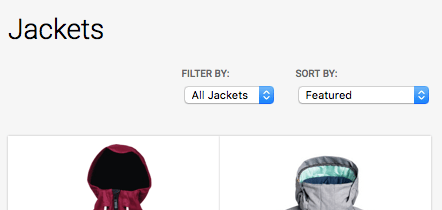
@media (max-width: 480px) {

.product-filter {

flex-direction: column;

}

}



Our header and filters no longer overlap, but we can still improve the layout by floating the filters to the left. Previously, we floated the elements to the right with the align-self: flex-end; property. Now, we’ll want to add align-self: flex-start;

@media (max-width: 480px) {

.product-filter {

flex-direction: column;

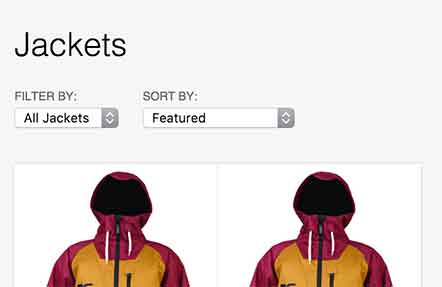
}

.sort {

align-self: flex-start;

}

}



And just like that, we now have a flexible and responsive layout for our products.

**Compatibility**

The biggest pushback with flexbox is always browser support. But as we mentioned earlier in this article, support is pretty good these days. Older IE that doesn’t support flexbox [isn’t even supported by Microsoft anymore](http://thenextweb.com/microsoft/2016/01/05/web-developers-rejoice-internet-explorer-8-9-and-10-die-on-tuesday/).

Like every web project you work on, you should always perform thorough testing to make sure that your visitors’ experience is optimized and that your layout meets your project’s requirements.

**Conclusion**

In the above tutorial, we built a powerful responsive layout for displaying a set of products using flexbox. Unlike other CSS methods not intended for building layouts, flexbox is a powerful tool focused on this goal, and you should take advantage of it. Flexbox layouts can make our sites and apps more flexible and resilient.