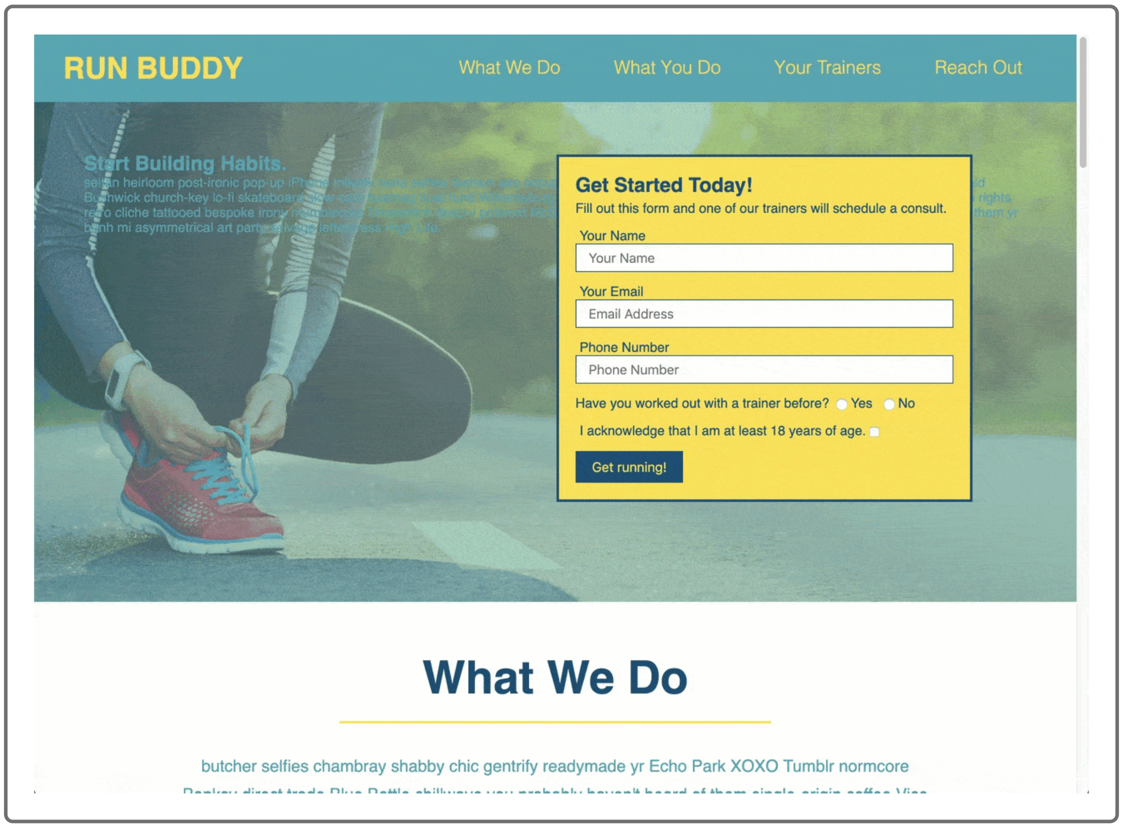
2.2.4 Add Flexbox to the Header

Our first task is to add flexbox to the header so it can accommodate various screen sizes gracefully. Before flexbox, CSS layout properties such as `float` were used to do this. As mobile devices became more popular, the inadequacies of `float` became apparent: content would often would break, overlap, or overflow off the screen. But it was the only viable option, so developers used it.

Flexbox (or flexible box model) was first introduced in 2009 but wasn't fully supported by browsers until 2017—and was it ever welcomed with open arms by the developer community! Developers finally had a tool that let them achieve modern layouts on any screen size.

The current Run Buddy site uses floats in the `<header>`. As you can see below, it doesn't resize very well:

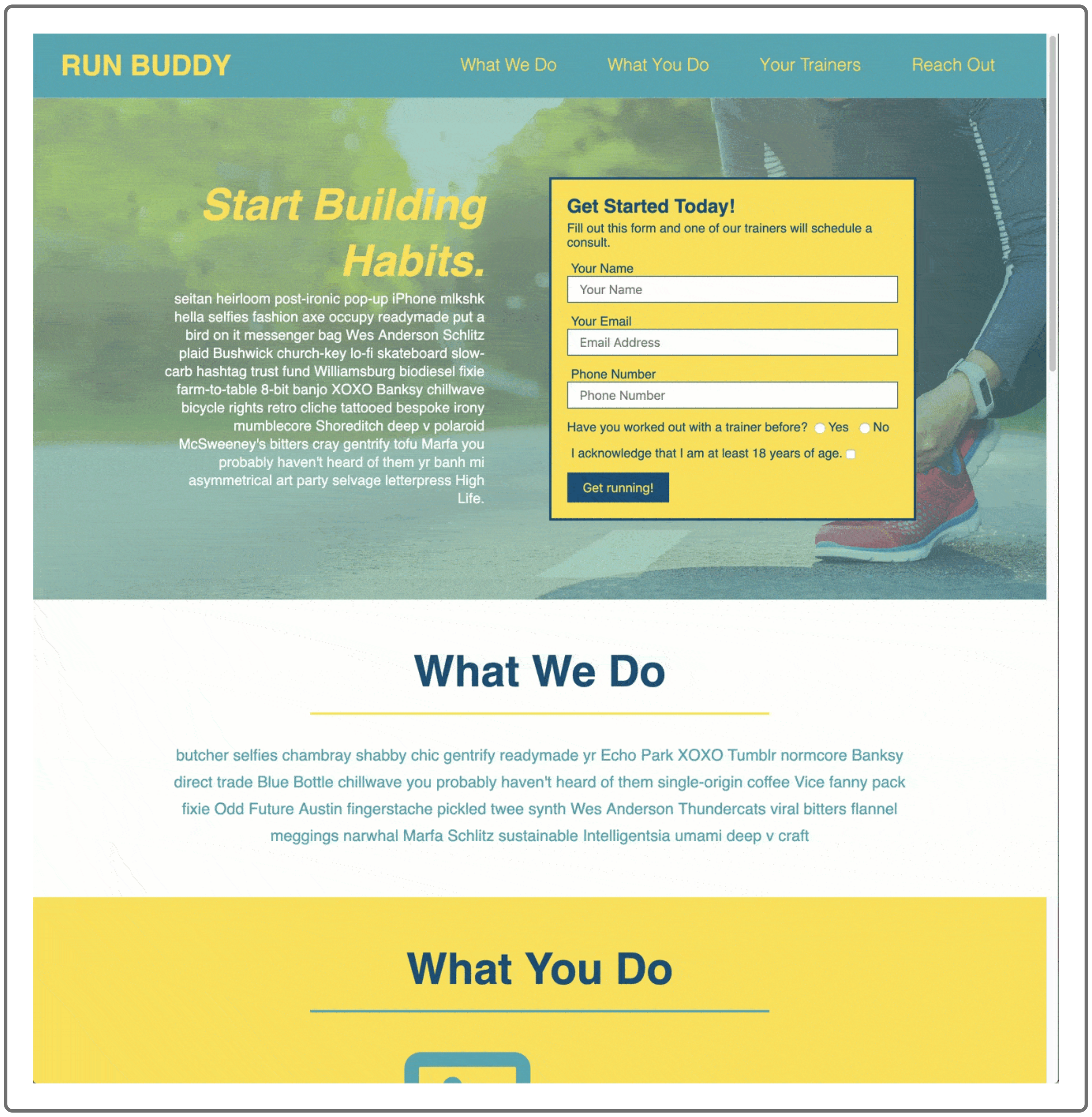


Get Started Today!
Fill out this form and one of our trainers will schedule a consult.
Your Name
Your Name
Your Email
Email Address
Phone Number
Phone Number
Have you worked out with a trainer before? Yes No
I acknowledge that I am at least 18 years of age.
Get running!

What We Do

butcher selfies chambray shabby chic gentrify readymade yr Echo Park XOXO Tumblr normcore

Here's how the header will resize after we add flexbox to it:



Start Building Habits.

seitan heirloom post-ironic pop-up iPhone mlkshk hella selfies fashion axe occupy readymade put a bird on it messenger bag Wes Anderson Schlitz plaid Bushwick church-key lo-fi skateboard slow-carb hashtag trust fund Williamsburg biodiesel fixie farm-to-table 8-bit banjo XOXO Banksy chillwave bicycle rights retro cliche tattooed bespoke irony mumblecore Shoreditch deep v polaroid McSweeney's bitters cray gentrify tofu Marfa you probably haven't heard of them yr banh mi asymmetrical art party selvage letterpress High Life.

Get Started Today!

Fill out this form and one of our trainers will schedule a consult.

Your Name: Your Name

Your Email: Email Address

Phone Number: Phone Number

Have you worked out with a trainer before? ○ Yes ○ No

I acknowledge that I am at least 18 years of age. ☐

Get running!

What We Do

butcher selfies chambray shabby chic gentrify readymade yr Echo Park XOXO Tumblr normcore Banksy direct trade Blue Bottle chillwave you probably haven't heard of them single-origin coffee Vice fanny pack fixie Odd Future Austin fingerstache pickled twee synth Wes Anderson Thundercats viral bitters flannel meggings narwhal Marfa Schlitz sustainable Intelligentsia umami deep v craft

What You Do

Flexbox creates HTML parent/child (or container/container-item) relationships. The parent keeps track of how much space each child takes up and repositions all of the children accordingly. This allows developers to focus only on the child elements' content.

Flexbox is quickly becoming the industry standard, so it's a great skill to add to your developer tool belt!

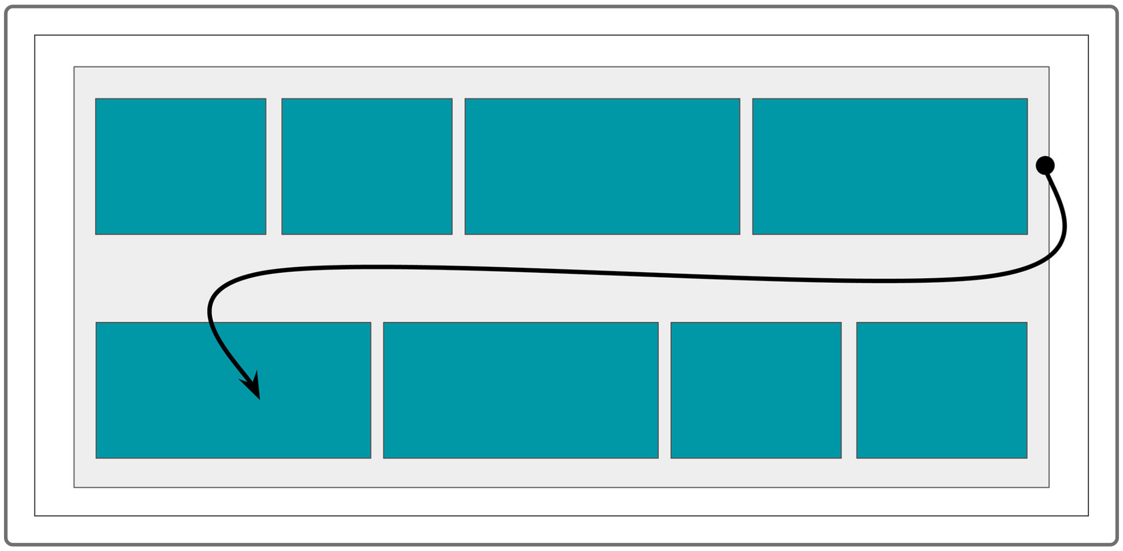
The Magic of Flexbox

When a container is given a `display` value of `flex`, it can create a **one-dimensional layout**, meaning it controls the distribution of its content along either a horizontal x-axis (row) or vertical y-axis (column).

By default, a flexbox is set to be a row, meaning it attempts to fit all of the child elements on one horizontal line until it hits the right edge of the parent. At that point, it has to decide whether to do one of the following:

- Make all of the child elements narrower to accommodate a new sibling on that horizontal line. This usually results in child elements looking "squished."
- Tell the flexbox parent that it's okay to overflow its content onto the next line. This is known as **wrapping**. This approach is commonly used in conjunction with flexbox containers to calculate smart layouts for various-sized screens.

This image shows how wrapping can work:



Another thing flexbox does well is horizontally or vertically aligning child elements. For example, it can be used to vertically center HTML elements in a container, which used to be nearly impossible with CSS. Flexbox is also great at evenly spacing elements by calculating the free space available and distributing it among child elements.

DEEP DIVE

DEEP DIVE

For more details, check out the **[MDN web docs on flexbox (https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox)]** .

Prepare the Header for Flexbox

Much of this lesson involves removing CSS properties and adding flexbox properties in their place. We'll start making the `<header>` element a flexbox by removing a few pieces from the CSS file:

- In `style.css`, remove the `display: inline` declaration from `header h1` so that it looks like this:

```
header h1 {
  font-weight: bold;
  margin: 0;
  font-size: 36px;
  color: #fce138;
}
```

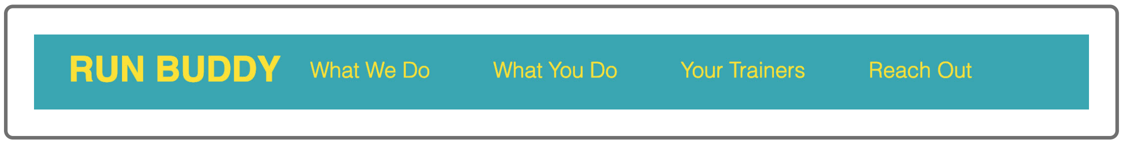
- Then remove the `float:right` declaration from `header nav`:

```
header nav {
  margin: 7px 0;
}
```

- Now apply flexbox by adding the following CSS property declaration to the `header` rule:

```
display: flex;
```

That's all that needs to be added to get our elements back on the same horizontal line! The header should now look like this:



The `display: flex` declaration makes the `<header>` a row by default, so there's no need to explicitly declare it. And as you can see, flexbox evenly distributed the space of the parent element (`<header>`) among the children elements (the `<h1>` and `<nav>` links). What makes this more interesting is that both of those children are block-level elements by default, and adding flexbox overrides that default behavior.

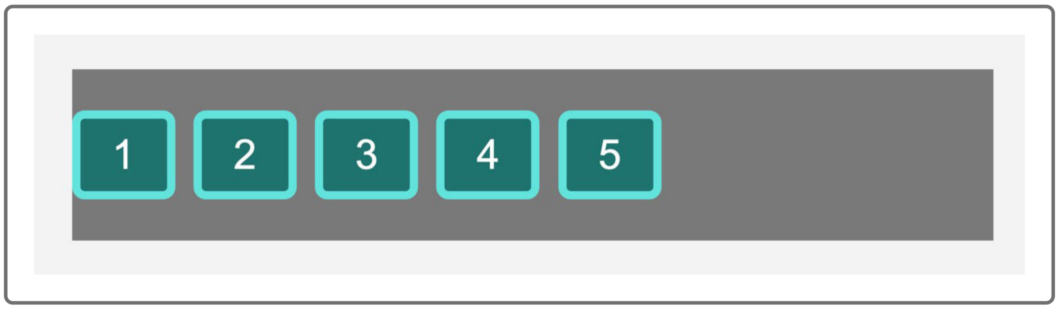
HIDE PRO TIP

To change a flexbox from a row to a column layout, you can use the `flex-direction` property with a value of `column`.

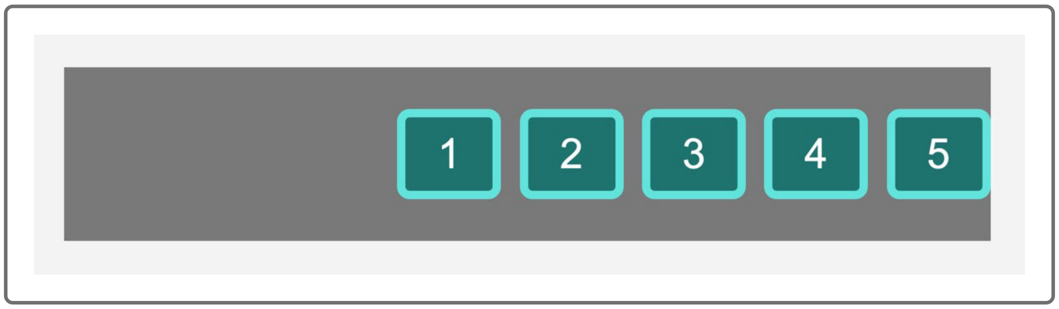
There are just a few more things to do to the header at this point. Did you notice that the navigation items seem to be skewed to the left, closer to the `<h1>`? We can adjust that by using a property called `justify-content`.

The `justify-content` property only applies to elements with a `display` property value of `flex` or `grid` (more on that later). It allows you to control spacing between child elements with the following values:

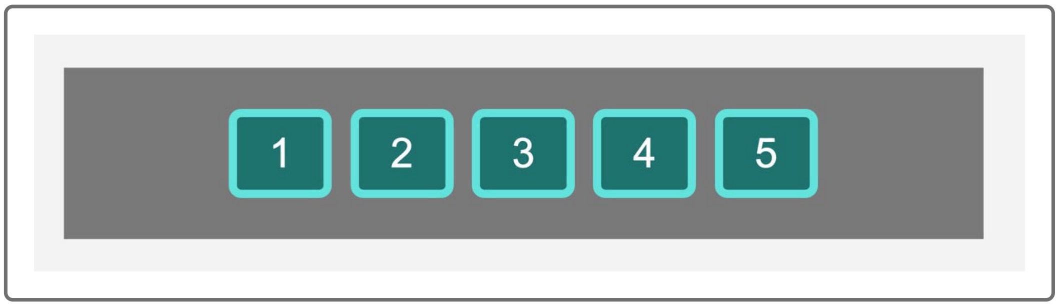
- `flex-start`: This left-justifies all of the elements in the container. This is the default.



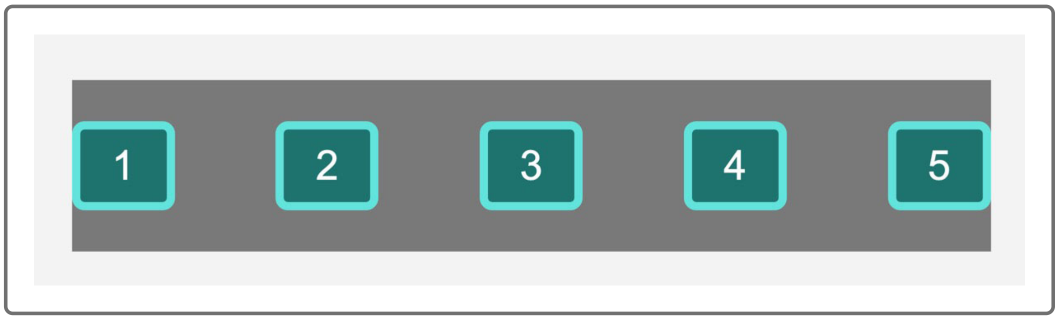
- `flex-end`: This right-justifies all of the elements in the container.



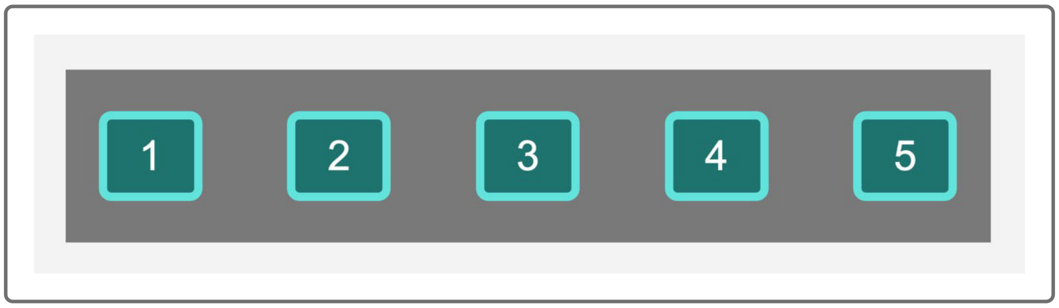
- `center`: Positions all elements as close to the center of the container as possible.



- `space-between`: Distributes all empty space evenly between child elements so they are evenly spaced apart.



- `space-around`: This is like `space-between`, but it also adds space between the elements and the edges of the container, so the left-most and right-most elements are not flush against those edges.



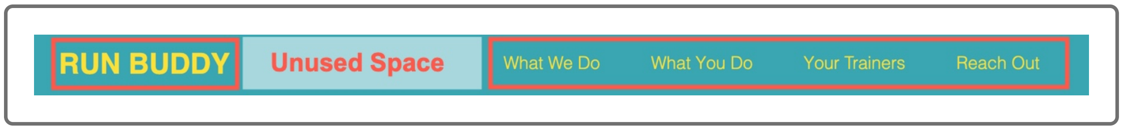
DEEP DIVE ▲

DEEP DIVE

To learn more, read the **MDN web docs on the justify-content property (https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content)**.

Which one of these is right for our layout? We currently have two child elements in the `<header>`: the `<h1>` and `<nav>` elements. We want them to be spaced apart as much as possible with the `<h1>` on the left and the `<nav>` on the right.

This image can help visualize what we need:



Let's add the following declaration to the `header` CSS rule:

```
justify-content: space-between;
```

This property declaration takes all of the unused space in the `<header>` and puts it between the two elements. It determines the unused space by looking at each child element and seeing how much the border-width, margin, padding, and content add up to. If these don't add up to 100% of the parent, then whatever is left over is considered the unused space.

PAUSE

What is it called when we calculate an element's dimensions by adding the content, padding, border-width, and margin values?

The CSS box model.

Hide Answer

So our `<header>` content looks great, but what happens when the screen gets smaller? Take a minute and resize your browser window to find out.

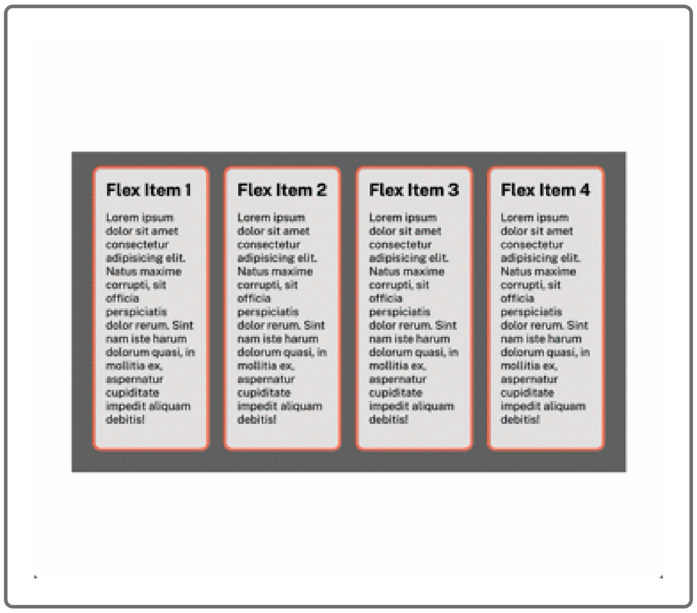
As you probably noticed, the content becomes as small and tight as possible or bursts out the right side of the container. Instead, we'll tell the container that when its child elements can't fit on one line, let them break onto the next line.

Well do this by adding a `flex-wrap` property, which allows the flexbox container to let its children wrap onto the next line. By default, the value of `flex-wrap` is set to `none`, so we need to explicitly tell it to be `wrap` by adding the following to the `header` CSS rule:

```
flex-wrap: wrap;
```

Now when we resize the browser window, the `<header>` will collapse onto a second line when it runs out of room.

The following animation demos a similar use for `flex-wrap`:



Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus maxime corrupti, sit officia perspiciatis dolor rerum. Sint nam iste harum dolorum quasi, in mollitia ex, aspernatur cupiditate impedit aliquam debitis!

Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus maxime corrupti, sit officia perspiciatis dolor rerum. Sint nam iste harum dolorum quasi, in mollitia ex, aspernatur cupiditate impedit aliquam debitis!

Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus maxime corrupti, sit officia perspiciatis dolor rerum. Sint nam iste harum dolorum quasi, in mollitia ex, aspernatur cupiditate impedit aliquam debitis!

Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus maxime corrupti, sit officia perspiciatis dolor rerum. Sint nam iste harum dolorum quasi, in mollitia ex, aspernatur cupiditate impedit aliquam debitis!

Now is a good time to save your progress with Git using the following commands:

```
git add .
git commit -m "add flexbox to header"
git push origin feature/flexbox
```