Adding a 'Loading' Element

Showing the rest of the list items -- infinitely. (Or almost)

WE'LL COVER THE FOLLOWING

- Changelog
- Client-side state handling

Changelog

The next thing we need to do is to fetch more tweets dynamically. This will involve a few different changes:

- We need to make some HTML elements that shows a loading state. To simplify our use case, we will once again use the beautiful stock dog instead of a spinning animation.
- We need to introduce a delay into our server, otherwise we can't test the loading state. We'll add a two-second pause.
- We need to detect when to load. There are global variables that let us access the user's current scroll position at any given time. We can fetch more when it's near the bottom of the page.
- We need to keep track of the last tweet ID so that we can start using it in subsequent calls.

Client-side state handling

Our API is already set up to handle asynchronicity with callbacks, so the introduction of a delay can be implemented with minimal changes:

```
function getFunction(url, data, callback) {
  const domain = url.substring(0, url.indexOf("/"));
  const endpoint = url.substring(url.indexOf("/"), url.length);
```

```
  setTimeout(() => callback(endpoints[endpoint]["get"](data)), 2000);
}
```

During the time in which the client has made the request, but the callback that handles the response has not yet been invoked, we should show the user something to let them know that more is coming.

Our component will now have to keep state. When the API request is made, the state is immediately set to 'loading,' and the loading element should show. In the callback function, we reset the state and replace the loading element with the content just fetched.

Because setting state has side effects, we want to have an abstraction like `setPending()` instead of doing both `state = 'loading'` and `addLoadingElement()`. That way, everywhere that we set the pending state, we just need to call one function, and any changes to actions that happen upon the pending state are restricted to one place.

```
const States = {
  PENDING: 'pending',
  READY: 'ready'
};

let componentState = States.READY;
const loadingElement = document.createElement('div');
// Give it the same style
loadingElement.classList.add('tweet');
loadingElement.innerHTML = `
  Here I am... Loading...
  <img class="loading__image" src="http://educative.io/udata/1m5lkL7p9Q0/dog.jpeg" />
`;

function setPending() {
  componentState = States.PENDING;
  document.body.appendChild(loadingElement);
}

function setReady() {
  componentState = States.READY;
  document.body.removeChild(loadingElement);
}
```

The states aren't exactly being used yet – but I know they'll be useful!

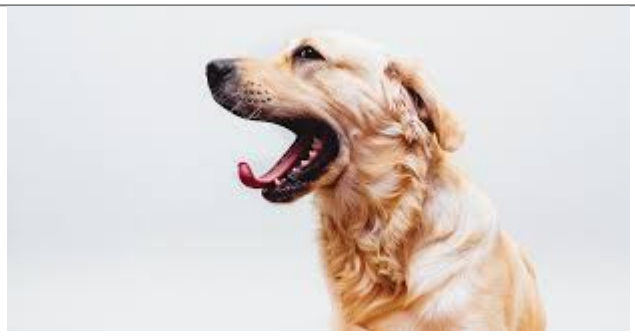
Here's our component so far. It's got a slight delay at the beginning, during which time it shows the loading element. If you missed it, just press the "HTML" tab and press back to "Output."

Output

JavaScript

HTML

CSS (SCSS)

Here I am... Loading... 

We’re now ready to load more than the first set of tweets. Let’s do that in the next lesson.