5.5.7 Customize Styles

The last thing the client wants to see in Taskmaster Pro is the addition of some custom CSS styles for various parts of the UI. Let's list them out so we don't lose track of them:

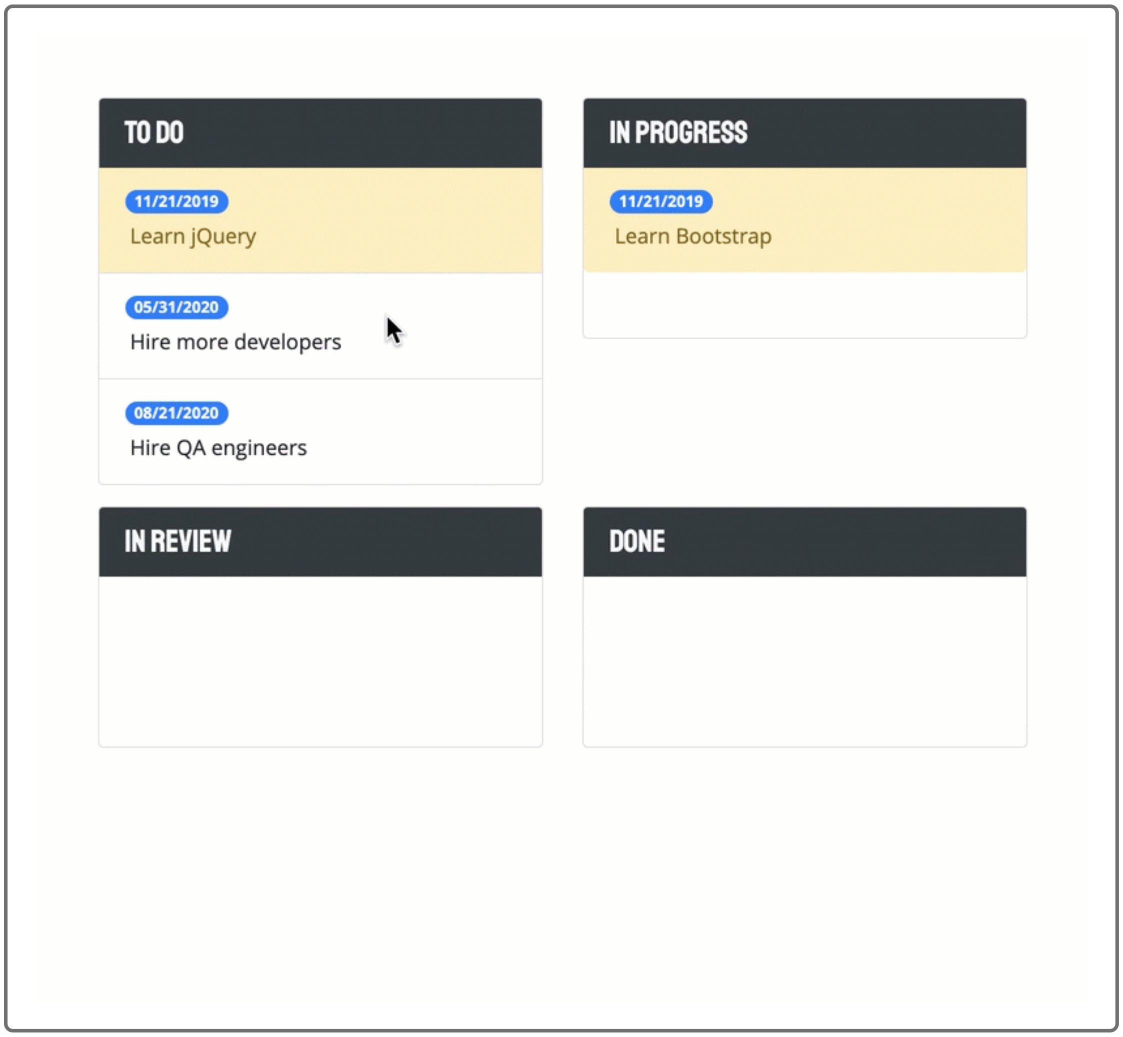
- When we currently drag a task, there's no indication of what area of the app is a dropzone, so we're going to highlight them when a drag event starts.
- The trash dropzone takes up a little too much space on the site. Since it doesn't need to be visible unless a task item is being dragged, we'll only have it show up at that time.
- The buttons in the left column and the modal form are very—as the client put it—"Bootstrap-y." We're going to update their color scheme so they don't feel so plain.

All three of these will involve the creation of custom CSS classes. Some of them will be easy to implement, as we can simply add the classes to the HTML elements. Others will involve using jQuery to dynamically add and remove classes during specific `.sortable()` and `.droppable()` events.

We've been able to rely on Bootstrap for so much of this project's design, but sometimes there's no substitute for custom CSS styles!

Highlight Task List Dropzones

The client envisions a task list looking like this animation shows when a task is dragged:



There are two different states we're going to need to account for here. First, we need a style that tells users what areas are dropzones. Then we'll need another style that tells users they're actively dragging a task over the list.

Let's start by creating the following two classes in the `style.css` file:

1. Create a CSS rule for a class called `dropover`.

- Give it a `background-color` value of `rgba(0,0,0,0.3)`

2. Create a CSS rule for a class called `dropover-active`.
   - Give it a `background-color` value of `rgba(0,0,0,0.3)`

PAUSE

What does the last number in `rgba()` mean?

It represents the alpha transparency value. 1 is fully visible and 0 is invisible, any decimal point number in between will increase or decrease that color's visibility and let the color behind it show through it.

Hide Answer

Next, we will update the `.sortable()` method's settings to apply the `.dropover` class to all of our task lists on drag start, but then also apply the `.dropover-active` class when a task is directly dragged over a specific list.

Let's update the `.sortable()` method's event callbacks in `script.js`:

1. In the `activate` method, use jQuery to select `this` and add the class of `dropover` to it.
2. In the `deactivate` method, use jQuery to select `this` and remove the class of `dropover` from it.
3. In the `over` method, use jQuery to select `event.target` and add the class of `dropover-active` to it.

4. In the `out` method, use jQuery to select `event.target` and remove the class of `dropover-active` from it.

HIDE HINT

We've used all of these jQuery methods throughout the application, so don't be afraid to look at the rest of the code for syntax help!

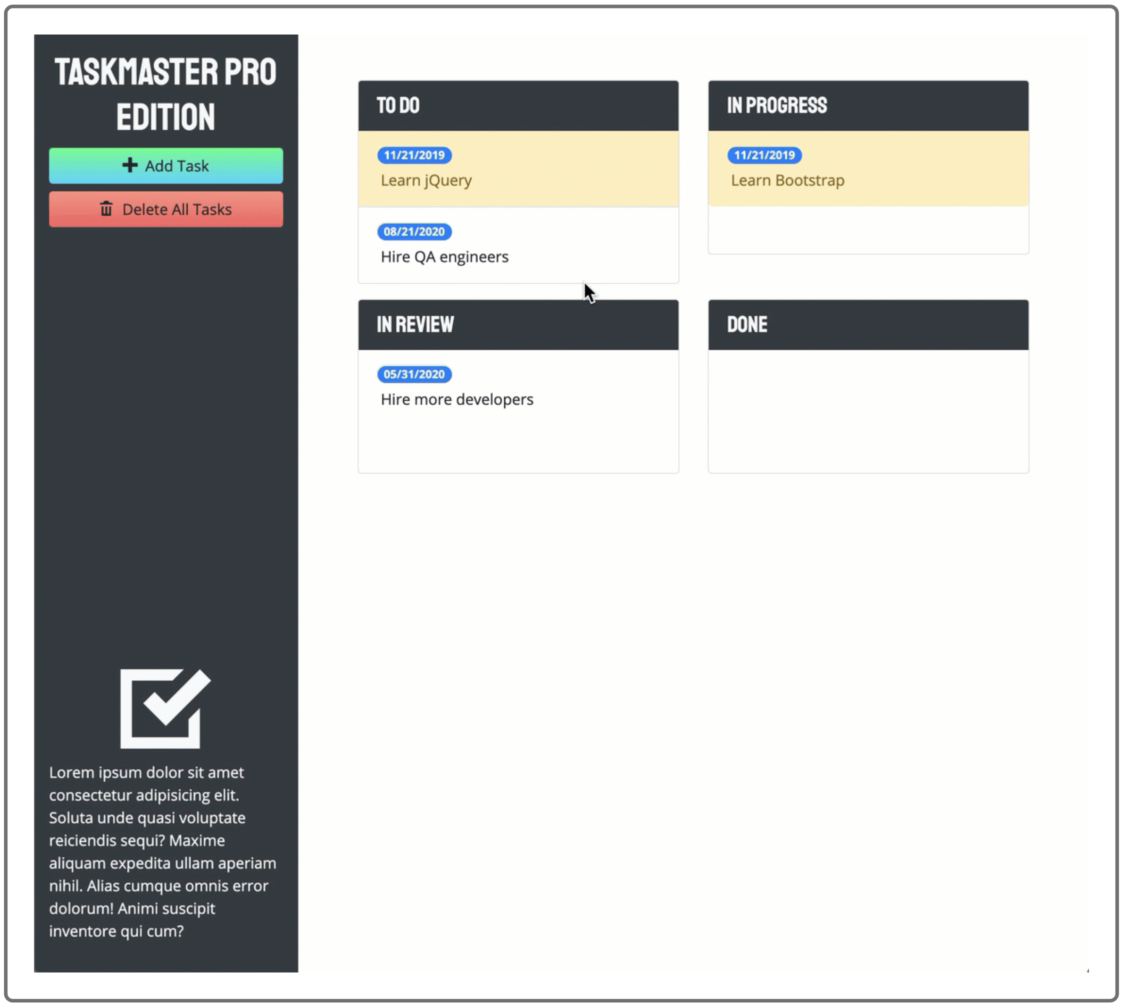
Save `script.js`, refresh the browser, and try to drag some task items around. We can see that they all receive the `dropover` class when it starts and the `dropover-active` class is added to the list that has an element dragged over it. When the dragging stops, all of the classes are removed!

This will help users unfamiliar with drag-and-drop features understand how to use the app, and it wasn't that much work on our end!

Animate the Trash Dropzone

The trash dropzone, as it currently stands, is somewhat confusing for users. There's this big banner at the bottom of the page that doesn't do anything until a task is dropped over it. Since its purpose is to only accept elements being dragged, why don't we set it up to come into view when that drag event starts?

The client thinks something like this animation depicts would do the trick:


TASKMASTER PRO EDITION
Add Task
Delete All Tasks
TO DO
11/21/2019
Learn jQuery
08/21/2020
Hire QA engineers
IN PROGRESS
11/21/2019
Learn Bootstrap
IN REVIEW
05/31/2020
Hire more developers
DONE
Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta unde quasi voluptate reiciendis sequi? Maxime aliquam expedita ullam aperiam nihil. Alias cumque omnis error dolorum! Animi suscipit inventore qui cum?

As we can see, when we start dragging a task item, the trash area will come into view to be dropped in. This way it's not an eyesore when we don't need to delete a task.

Let's start in `index.html` and make a couple of adjustments to this element:

```
<div class="w-100 bg-danger p-3 text-center text-light display-4">
  <span class="oi oi-trash"> </span> Drop Here to Remove.
</div>
```

1. On the `<div>`, remove the classes for `display-4`, `bg-danger`, and `text-light` as we will use a custom styles instead of Bootstrap ones.

2. Also on the `<div>`, add a class called `bottom-trash`, as we will create a CSS rule for it next.

3. On the `<span>`, add these Bootstrap classes: `mx-auto` and `mb-4`.

Once that's all set, let's add three new CSS classes for the trash zone:

1. Create a CSS rule for the `bottom-trash` class with the following properties:
   - Set the `opacity` property to `0`
   - Set the `color` property to `#f1f1f1`
   - Set the `font-size` property to `2.3rem`
   - Set the `font-family` property to `"Staatliches", sans-serif`
   - Set the `transform` property to `translateY(150px)`
   - Set the `transition` property to `all .5s ease-in-out`
   - Set the `background-color` property to `rgba(234,0,39,0.7)`
2. Create a CSS rule for a class called `bottom-trash-drag` with the following properties:
   - Set the `transform` property to `translateY(0)`
   - Set the `opacity` property to `1`
3. Create a CSS rule for a class called `bottom-trash-active` with the following property:
   - Set the `background-color` property to `rgba(234,0,39,1)`

IMPORTANT

> Don't forget to look up any of these properties using Google or the MDN web docs if you're unfamiliar with how they're used!

Great, so now we have three classes set up for our trash zone. The `bottom-trash` class will give the dropzone all of its base styles and move it down 150px using the `translateY()` CSS function. Then `bottom-trash-drag` will be added to the trash dropzone element when a task is dragged, which will move the element back into view. Lastly, the `bottom-trash-active` class will be applied when a task item is dragged over the trash dropzone.

Let's go back to the JavaScript side of things and add or remove these classes when needed:

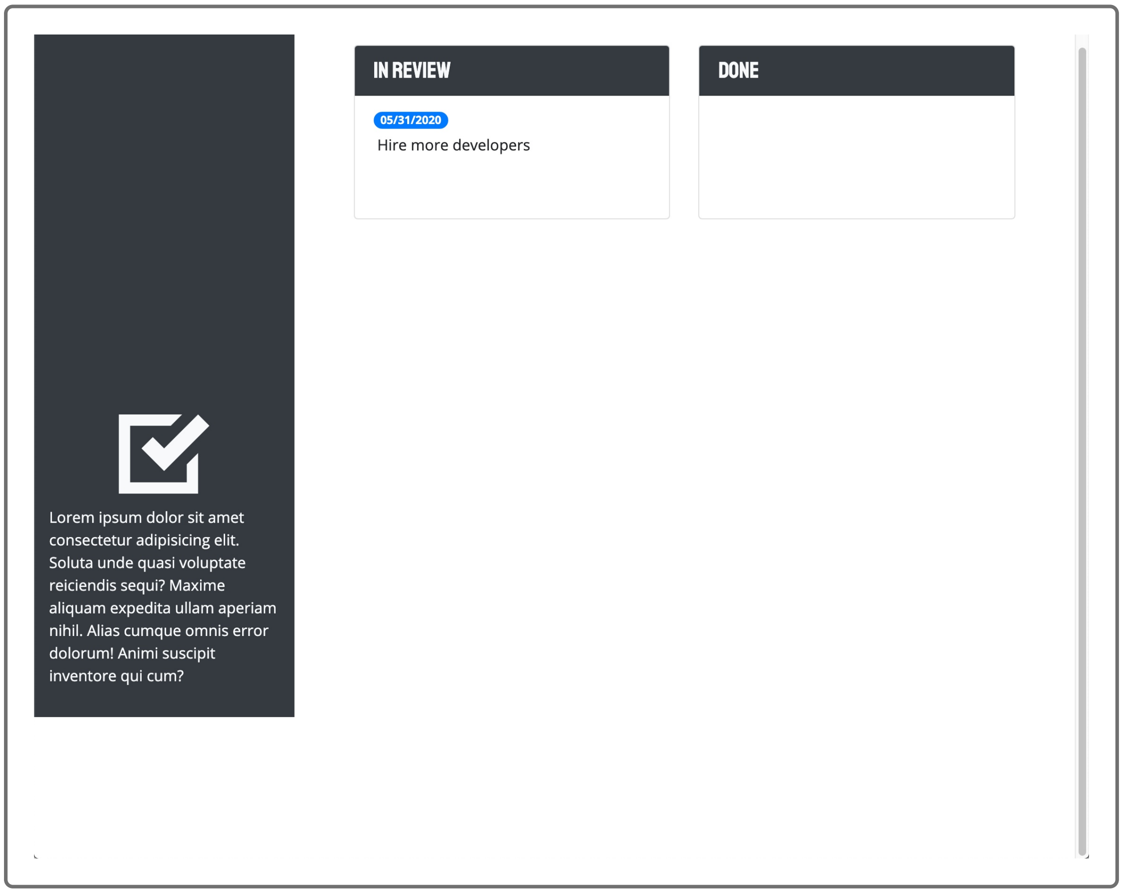
1. In the `.sortable()` method's `activate` method, use jQuery to select `.bottom-trash` and add the `bottom-trash-drag` class to it.

2. In the `.sortable()` method's `deactivate` method, use jQuery to select `.bottom-trash` and remove the `bottom-trash-drag` class from it.

3. In the `.droppable()` method's `over` method, use jQuery to select `.bottom-trash` and add the `bottom-trash-active` class to it.

4. In the `.droppable()` method's `drop` and `out` methods, use jQuery to select `.bottom-trash` and remove the `bottom-trash-active` class from it.

If we save `script.js` and refresh the page, we'll notice the trash isn't visible but the page overflows on the bottom, like this screenshot shows (you may have to scroll down the page to see this):


IN REVIEW
05/31/2020
Hire more developers
DONE
Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta unde quasi voluptate reiciendis sequi? Maxime aliquam expedita ullam aperiam nihil. Alias cumque omnis error dolorum! Animi suscipit inventore qui cum?

Let's fix this by finding the `<div>` element with an `id` of `trash`, and add the Bootstrap class `overflow-hidden` to it. (See the Bootstrap docs for how it works.) It should now look like this:

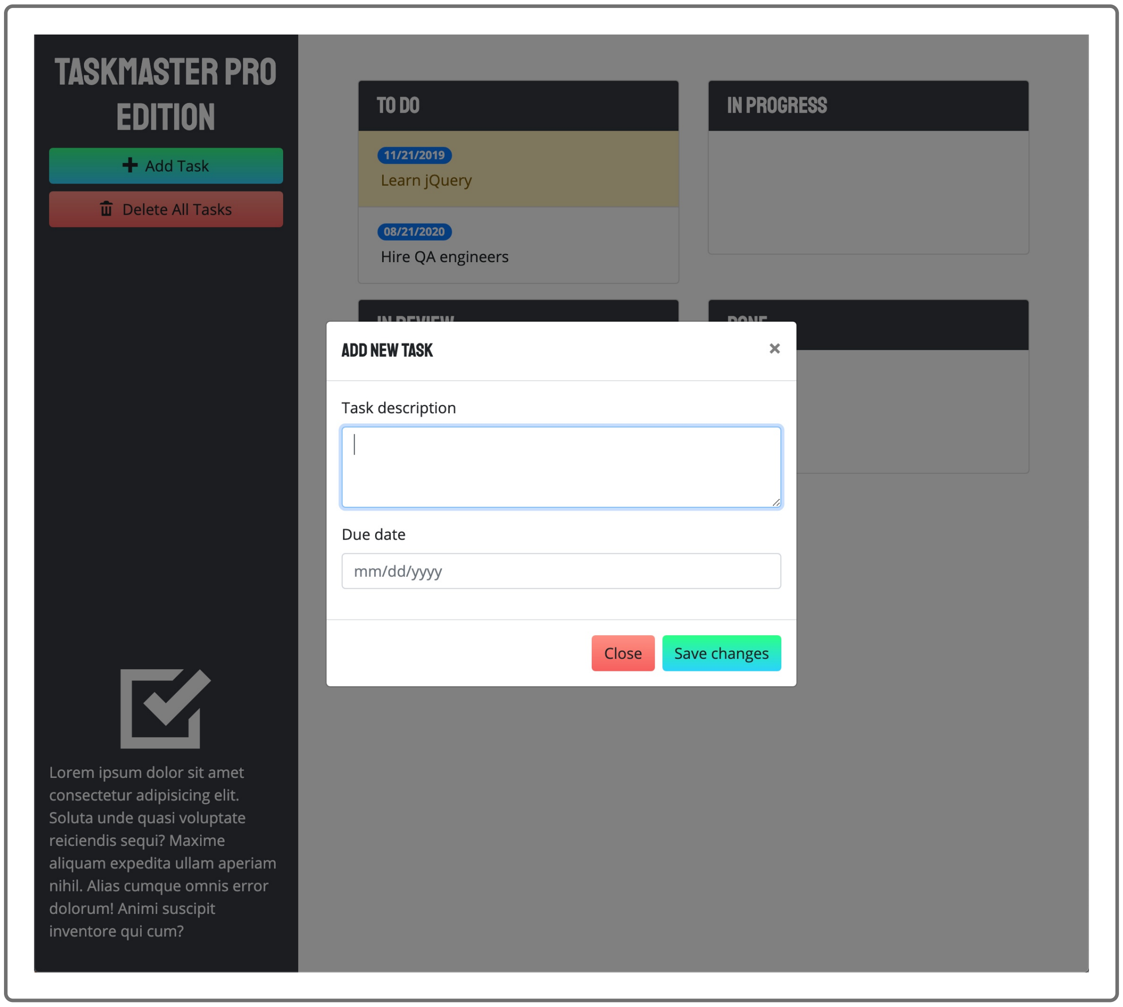
```
<div class="mt-auto overflow-hidden" id="trash">
  <div class="w-100 p-3 text-center bottom-trash">
    <span class="oi oi-trash"> </span> Drop Here to Remove.
  </div>
</div>
```

When we save `index.html` and refresh the browser, we'll see that there is no overflowing space anymore. This happened because we added the .bottom-trash class to the `<div>`. The transform setting on that class translates the Y position of the `<div>` down by 150px from its origin (`transform: translateY(150px)`).

Now when we try to drag a task item, we should see the trash dropzone slide in from the bottom and turn a little bit darker when we drag a trash item over it. This is a big win for us! We maintained the app's functionality while making a big improvement to the UI, which is not always an easy task. Luckily, with the tools at our disposal and knowledge in advanced CSS topics, we were able to get it done well!

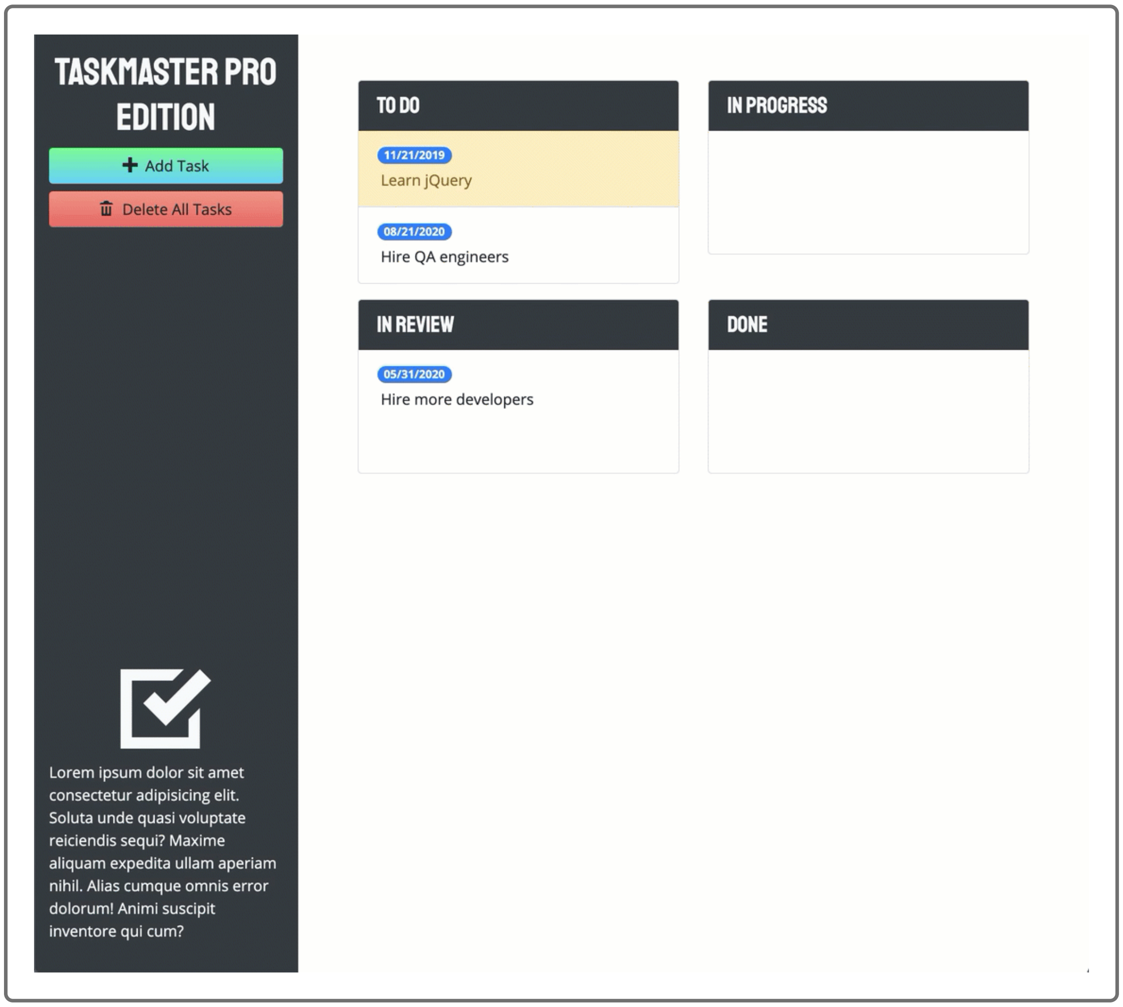
Customize Buttons

Let's take one more look at how the client wants the buttons to look on the page for both the left column and the modal:



Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta unde quasi voluptate reiciendis sequi? Maxime aliquam expedita ullam aperiam nihil. Alias cumque omnis error dolorum! Animi suscipit inventore qui cum?

So it seems the buttons for adding a new task will have a lime-green-to-teal gradient and the buttons for deleting all tasks and closing the model will have a light-red-to-red gradient. This seems like it could be pretty quick, but the client also thinks that gradient should change when the user points their cursor at it, like this animation shows:


TASKMASTER PRO EDITION
Add Task
Delete All Tasks
TO DO
11/21/2019
Learn jQuery
08/21/2020
Hire QA engineers
IN PROGRESS
IN REVIEW
05/31/2020
Hire more developers
DONE
Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta unde quasi voluptate reiciendis sequi? Maxime aliquam expedita ullam aperiam nihil. Alias cumque omnis error dolorum! Animi suscipit inventore qui cum?

The gradients shift upward when a cursor is over it, one turning more purple and the other a deeper shade of red. This may seem difficult, but you know all of the tools needed to achieve this, so let's get started!

We'll start by removing Bootstrap classes from HTML elements in `index.html`:

1. Find the `<button>` element in the `<header>` with a class of `btn-success` and replace it with `btn-add`.

2. Find the `<button>` element in `<header>` with a class of `btn-danger` and replace it with `btn-delete`.

3. Find the `<button>` element in `<div class="modal">` element with a class of `btn-secondary` and replace it with `btn-close`.

4. Find the `<button>` element in `<div class="modal">` with a class of `btn-primary` and replace it with `btn-save`.

Before you add these custom styles, refresh the page and make sure the functionality still works. You should first notice that the button background colors are removed. But if you click on the Add Task button, the modal still appears, but its Save button doesn't work anymore!

This is because we removed the class `btn-primary` from it, and we're using that class to target the button in our `script.js` file. Let's find what uses it in `script.js` and change it:

- Find the jQuery selector that has `btn-primary` in it and replace it with `btn-save`.

HIDE PRO TIP

Don't forget to use the "Find" tool in Visual Studio Code!

Once we find and fix that selector, try it again and make sure it's working.

Now let's create our CSS rules:

1. Create a CSS rule that selects both `btn-add` and `btn-save` and give it the following CSS properties:
   - Set `background-color` to `#FA8BFF`
   - Set `background-image` to `linear-gradient(45deg, #FA8BFF 0%, #2BD2FF 52%, #2BFF88 90%)`

- Set `transition` to `all .5s ease-out`
- Set `background-size` to `1px 100px`
- Set `background-position` to `0 -10px`

2. Create a CSS rule that selects `btn-close` and `btn-delete` and give it the following CSS properties:
   - Set `background-color` to `#EA0028`
   - Set `background-image` to `linear-gradient(180deg, #FF9A8B 0%, #EA0028 100%)`
   - Set `transition` to `all .5s ease-out`
   - Set `background-size` to `1px 100px`
   - Set `background-position` to `0 -10px`
3. Create a CSS rule for the all of the above classes, but this time for their `:hover` state:
   - Set `background-position` to `1px -50px`

HIDE HINT

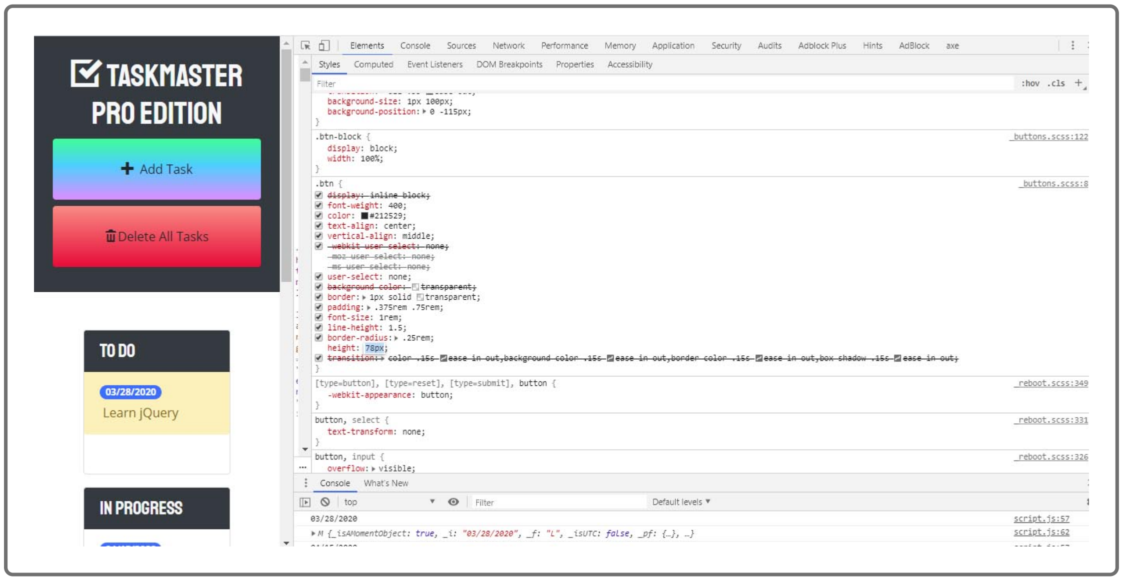
To select multiple elements, use a comma to separate them like this:

```
.class-one:hover,
.class-two:hover,
.class-three:hover {
  /* Styles go in here */
}
```

Now when we save and refresh the page, our new button styles will be in place! There are some interesting uses of the CSS background properties going on here, so let's look at those more closely.

We set the background to have a gradient, and if the browser doesn't support it, the color will fall back to whatever `background-color` is using. We also used the `background-size` property to make that gradient a lot taller (100px) than the actual button height (38px), so that's why we don't see the purple-ish color (#FA8BFF) from the gradient on load.

To see how the button is showing just a section of the background color, right-click the button in the browser to bring it up in the DOM in the developer's tools. Then locate the `.btn` style rule in the styles, and add `height: 38px` to that style rule. (You've just changed the style dynamically! You can do this with any element.) The button should be the same size. But click on "38px" and the field will become editable. Press on the up and down arrow keys on your keyboard, and that value will go up and down. The height of the button will alter with the changes to the setting. You'll then see how the displayed background is part of a larger, 100px background. Your interaction with the style rules and the browser display should look something like this image:


TASKMASTER
PRO EDITION
Add Task
Delete All Tasks
TO DO
03/28/2020
Learn jQuery
IN PROGRESS
Elements Console Sources Network Performance Memory Application Security Audits Adblock Plus Hints AdBlock axe
Styles Computed Event Listeners DOM Breakpoints Properties Accessibility
Filter :hov .cls +

```
  background-size: 1px 100px;
  background-position: ▸ 0 -115px;
}
.btn-block {                                                    buttons.scss:122
  display: block;
  width: 100%;
}
.btn {                                                          buttons.scss:8
☑ display: inline-block;
☑ font-weight: 400;
☑ color: ■#212529;
☑ text-align: center;
☑ vertical-align: middle;
☑ -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
☑ user-select: none;
☑ background-color: □transparent;
☑ border: ▸ 1px solid □transparent;
☑ padding: ▸ .375rem .75rem;
☑ font-size: 1rem;
☑ line-height: 1.5;
☑ border-radius: ▸ .25rem;
  height: 78px;
☑ transition: ▸ color .15s ease-in-out,background-color .15s ease-in-out,border-color .15s ease-in-out,box-shadow .15s ease-in-out;
}
[type=button], [type=reset], [type=submit], button {            reboot.scss:349
  -webkit-appearance: button;
}
button, select {                                                reboot.scss:331
  text-transform: none;
}
button, input {                                                 reboot.scss:326
  overflow: ▸ visible;
```

```
03/28/2020                                                      script.js:57
▸ M {_isAMomentObject: true, _i: "03/28/2020", _f: "L", _isUTC: false, _pf: {…}, …}   script.js:62
```

We then set the `background-position` to move from -10px to -50px on hover, effectively moving the colored background up and down. Use the same process described above to edit the background position dynamically in the DOM to see the shift.

We've also used a `transition` property, which provides a nice sliding effect. If we didn't use the transition, the change would be immediate and jarring. You might strike the transition in the style rules, too, to see what the effect is like without the transition. (In case you're worrying, none of the changes you're making in the DOM will be permanent. Once you refresh the browser, the original CSS rules will be reinstated.)

Great job on finishing yet another great looking application! The additions from this last lesson may not do much in terms of core functionality, but they really add to the overall experience a user gets when they use the application. Let's wrap up this last feature branch and get Taskmaster Pro out to the client!