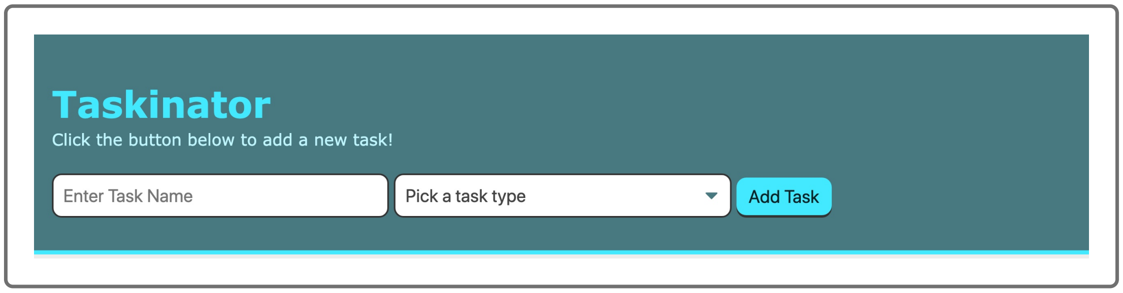
4.2.4 Add Task Form to HTML

We're very close to being able to dynamically create new tasks, so we don't need to hardcode test tasks anymore. Delete the `<li>` element with "A task is going to be added soon!" as its text. The parent `<ul>` element should now look like the following:

```
<ul class="task-list" id="tasks-to-do"></ul>
```

To create new tasks with unique content, we'll first need to implement an HTML form. Keep in mind that creating the form in HTML will only allow us to enter content to be submitted; capturing that content will be done in JavaScript right after.

Before discussing how this works, let's focus on setting up the form to look like this:



Currently we just have a button that triggers an event to create a new task, but we'll add form elements to go along with that button. Let's start by adding a `<form>` element in the `<header>`.

Let's replace the existing `<button>` with the following code. Update the `<header>` element to look like this under the `<p>` element:

```
<form id="task-form">
  <div class="form-group">
    <button class="btn" id="save-task" type="submit">Add Task</button>
  </div>
</form>
```

Obviously we aren't done, as we don't have the other two form elements yet, but let's take a moment and review a few things about the form elements we just added:

- We gave the `<form>` element an `id` attribute. This won't come into play just yet, but considering what we learned in the last lesson, we can assume that we'll use this attribute in the JavaScript code.
- We also wrapped the `<button>` element in a `<div>` with a class of `form-group`. We're going to wrap all of our form elements with this `<div>` to make styling it easier.

REWIND

Remember that it can be hard to control the layout of HTML elements designed for content, such as `<p>`, `<a>`, `<button>`, and `<input>`. To make things a bit easier, wrap HTML elements in container elements whose sole purpose is to handle where they go on the page.

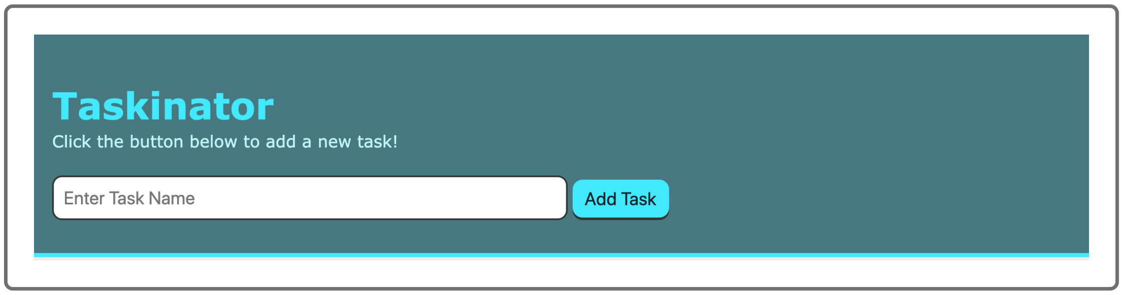
Add Task Input

Even after adding the `<form>` and `<div>` elements around the `<button>`, the header should still look almost the same. But now it's set up for us to add more content, so let's add an `<input>` element to capture our task's name.

Add the following right after the opening `<form>` tag:

```
<div class="form-group">
  <input type="text" name="task-name" class="text-input" placeholder="
</div>
```

Save `index.html` and refresh the page in the browser. You should see something like this image:



By adding the class of `text-input` to the `<input>` element, we overrode all of the browser's default styles for the element and made it align more with our app's style.

PAUSE

Where does the class `text-input` come from?

It comes from the style sheet that was provided to us for this project!

Hide Answer

Add Task Type Dropdown

Now we can write in a task name, but we're going to add one more form element: something to help us track what type of project it is. We'll use an HTML element called `<select>`.

Between the `<div>` elements that hold the text input and the button, add this HTML:

```
<div class="form-group">
  <select name="task-type">
    <option value="" disabled selected>Pick a task type</option>
    <option value="Print">Print</option>
    <option value="Web">Web</option>
    <option value="Mobile">Mobile</option>
  </select>
</div>
```

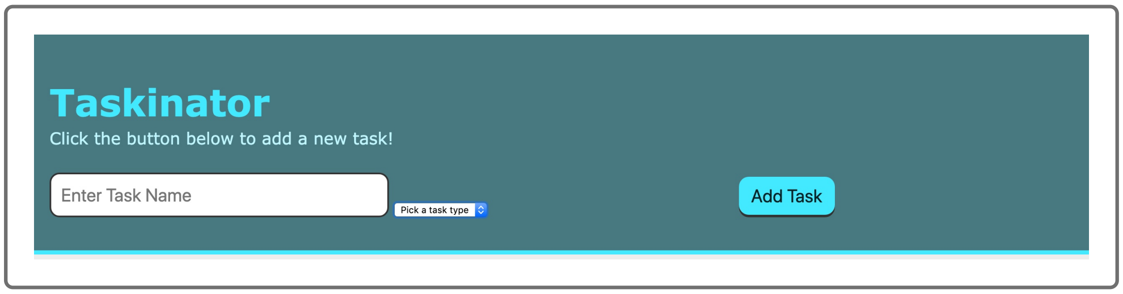
Save `index.html` and refresh the browser. We now have a form element that allows us to select from a predetermined list of options. We've never come across this type of form element before, so let's review a few points:

- We use `<select>` to tell the browser we're about to create this type of drop-down list, but we use `<option>` elements between the `<select>` tags to create the choices for that dropdown.

- We use the `name` attribute to help identify the form input. We use it with the `querySelector()` method in the JavaScript.
- A `value` attribute should accompany every option, as this value will be used in the JavaScript later to read the option we've picked.
- Adding the `disabled` and `selected` properties to the first option makes that option show up first but also applies a style that tells the user that this option isn't valid.

We used the options of "Print," "Web," and "Mobile" here because those are common project types we may be working on, but since this is a personal project, we can customize those options to fit our needs.

This new element looks pretty bad compared to the other form elements on the page, as this image shows us:

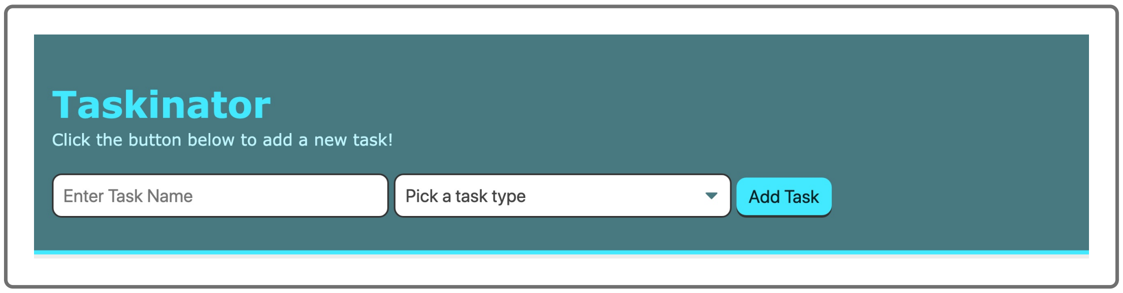


Elements like this, along with radio and checkbox inputs, get a lot of default styling from the browser, whether we like it or not. Luckily, the style sheet we're using has a CSS class ready for us, so let's implement it!

Add the following class to the `<select>` element to look like this:

```
<select name="task-type" class="select-dropdown">
```

If we save the file and refresh the browser, it should look a lot better, like this image:



We won't go too deep into the magic behind what makes this look good, but feel free to inspect it in Chrome DevTools. The key to styling an element like this is using the `appearance` CSS property, which tells a browser how to interpret the element. In this case, we're using that property to tell the browser to do absolutely nothing with the element and we'll style it ourselves.

The form is all set! Now that we have the HTML in place, let's jump back into the JavaScript file and make it functional.

Don't forget to add, commit, and push the feature branch.

DEEP DIVE

DEEP DIVE

To learn more, see the **MDN web docs on the select element. (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select)**