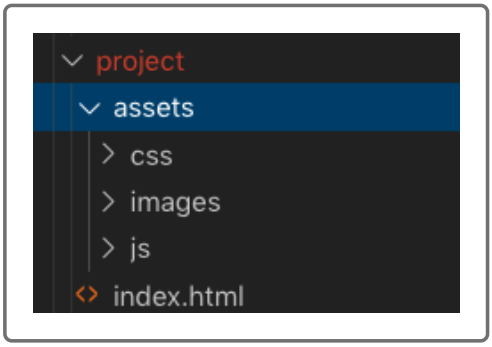
4.1.5 Build the HTML and CSS

First, **click here to download a zip file (https://static-fullstack-bootcamp.s3-us-west-1.amazonaws.com/module-4/module-4-assets.zip)** with the style sheet and an image we'll use for the project. When you download it, double click the `.zip` file and it'll unpack the two files we need so you can place them into their respective directories.

Then create the following folders and files and set up the basic structure of the project:

- `index.html`
- `assets` folder
- `css` folder inside `assets` folder
- `js` folder inside `assets` folder
- `script.js` inside `js` folder
- `images` folder inside `assets` folder

Move the `style.css` and image file to their proper folders (`css` and `images`, respectively), as shown in the below image:



Next, we need to add HTML to the `index.html` file. Luckily, there's a tool called Emmet that will create the initial boilerplate HTML for us. In VS Code, in the `index.html` file, type `html:5` and press Enter. This automatically creates the basic structure of an HTML5 file, saving us a lot of typing!

DEEP DIVE

Let's change a few elements on the HTML page to link the project files. Edit the `<link>` element, add the `<script>` element, and change the title so that your `index.html` file looks like this:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Taskinator</title>
  <link rel="stylesheet" href="./assets/css/style.css">
</head>

<body>

  <script src="./assets/js/script.js"></script>
</body>
```

```
</html>
```

Add the HTML

Let's look at the mock-up again:

Based on what the mock-up shows us, how many main blocks of HTML do we need? The use of background colors in the design can give us the answer. Let's dissect it:

- The block at the top with the dark-teal background holds the title of the application and a little more information about it, so we can use a `<header>` element there.
- Because the middle block with the gray background will hold the most important content for the application, we can use the aptly named `<main>` element there.
- The block at the bottom of the page will use the `<footer>` element.

So now that we have the three content sections identified, let's start adding them one by one, starting with the `<header>` element.

Create the `<header>` element inside the opening `<body>` element so that it looks like this:

```
<body>
  <header>

  </header>

  <script src="./assets/js/script.js"></script>
</body>
```

Now that we have the initial `<header>` element set up, let's add a few more elements to complete this section:

1. Add an `<h1>` element called `Taskinator`, with a class attribute value of `page-title`.

2. Add a `<p>` element with the following text: `Click the button below to add a new task!`

3. Lastly, add a `<button>` element called `Add Task`, with a class of `btn`.

After adding these elements to the `<header>`, the `index.html` file should look like this:

```
<header>
  <h1 class="page-title">Taskinator</h1>
  <p>Click the button below to add a new task!</p>
  <button class="btn" >Add Task</button>
</header>
```

Save the `index.html` file and open it in the browser. You should see something like the following image:

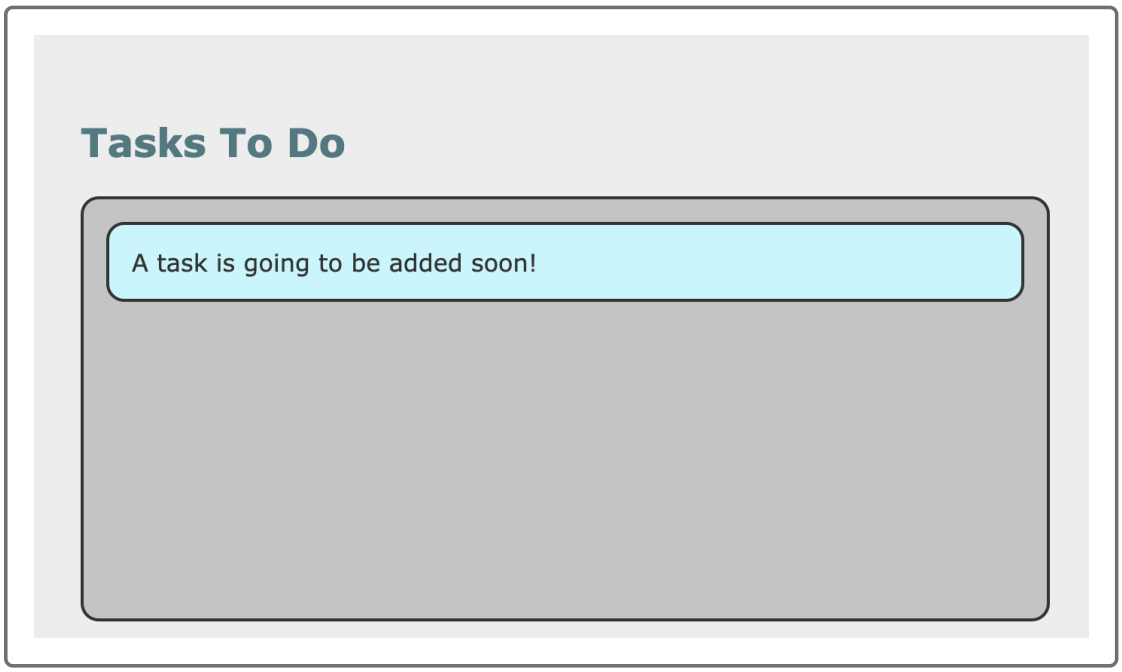
Wow, that looks pretty good considering we haven't written any new CSS yet. Remember that `style.css` file you downloaded and put into the `css` folder earlier? That contained all the CSS for the page. All you needed to do was match the class selectors in the `style.css` file to the class attributes of the HTML elements!

ON THE JOB

Development teams often lack the time or bandwidth to create new, extensive style sheets for every project, much less every webpage. Software companies and agencies often have a style sheet designed by a UX/UI team to keep the styles or branding consistent across the website, which dev teams can leverage by adding class attributes to the HTML elements.

Now let's proceed with the next block of HTML content—the `<main>` element, which will contain the task list. Add a `<main>` element to your code, giving it a class attribute of `page-content`.

To determine the next step, let's look at the finished application's `<main>` element, as shown in the image below:



You should see three elements in this image:

- Heading
- Task item
- Task list

Because we need the task list heading and the task list itself to always render together, let's wrap them in a `<section>` element.

Now let's add these elements to the HTML inside the `<main>` element:

- A `<section>` element with the class attribute `task-list-wrapper`, and the following elements nested inside:
    - An `<h2>` element with the class attribute `list-title`, called "Tasks To Do"
    - A `<ul>` element with the class attribute `task-list`
    - Nested in the `<ul>` element is an `<li>` with class attribute `task-item`, saying something like "A task is going to be added soon!"

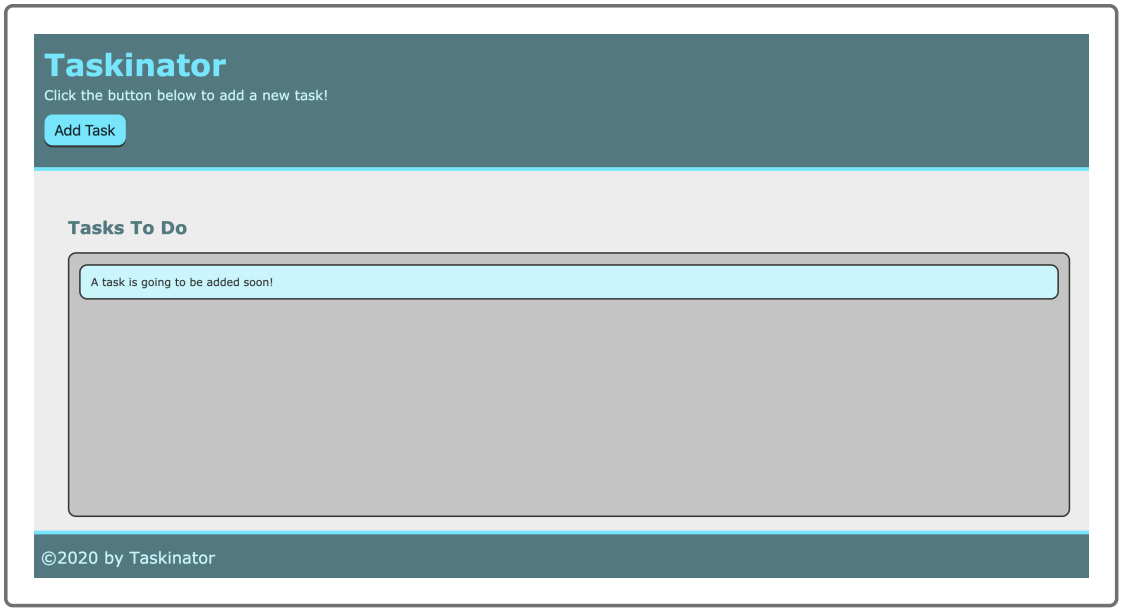
The finished code should look like this:

```
<main class="page-content">
  <section class="task-list-wrapper">
    <h2 class="list-title">Tasks To Do</h2>
    <ul class="task-list">
      <li class="task-item">A task is going to be added soon!</li>
    </ul>
  </section>
</main>
```

To finish the HTML, let's also add the `<footer>`, which contains the copyright information:

```
<footer>
  &copy;2020 by Taskinator
</footer>
```

Time to save and open the `index.html` in the browser! Compare your page to the following mock-up to see if you've missed anything:



Nice work! Add and commit, then push the feature branch up to GitHub.

DEEP DIVE