11.6.1 Bootstrap Components

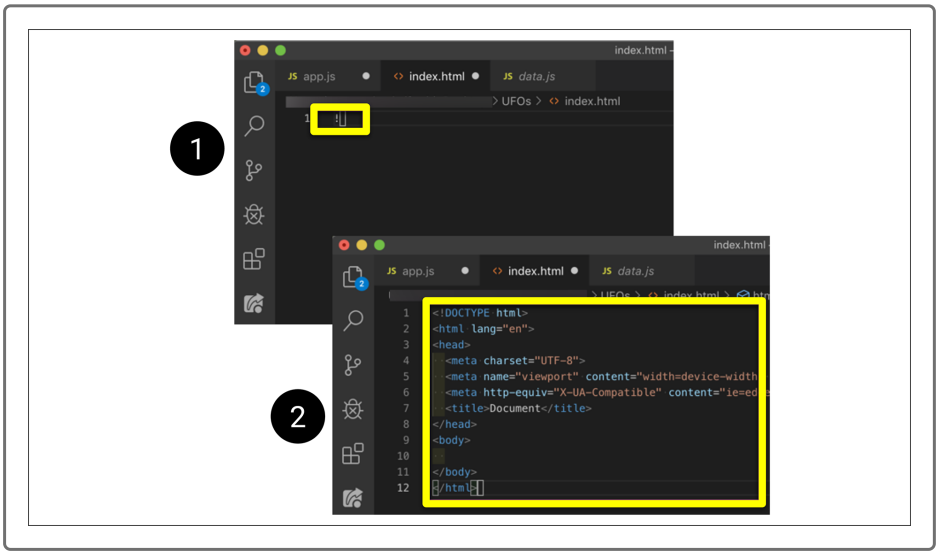
With our help, Dana has assembled code that will not only build a functional table but will add the ability to filter it as well. That's definitely an accomplishment—JavaScript isn't an easy language to get the hang of. But even though she thinks her code is structurally sound and will work without a hitch, she still needs to test it. To do so, it's time to build the HTML page Dana will be showcasing her work on.

She'll need to keep the different ids she created in app.js in mind while she pieces this webpage together. There will be a button (`#filter-btn`) somewhere as well as a `#datetime` id nestled into some tags. She'll need to build the base of the table so that the code we helped construct knows where to put the table's data. This means that the columns and rows will all need to be defined manually.

Thankfully, Dana still has the storyboard she assembled earlier, which will speed up the assembling process.

The time has come to build the webpage. This is known territory for Dana, so she's excited to take a short brain break and work with the more familiar tools of Bootstrap and HTML. It's also a test to see if the JavaScript code is working.

First, we'll need to go back to the `index.html` file that we created earlier. If you haven't already, open that file in VS Code. Next, use a shortcut to autofill the basic HTML layout by typing an exclamation mark on the first line, then press enter on your keyboard.



With the basics already completed for us, we can start customizing. First, we can change the title of the document to "UFO Finder."

Continue the setup by adding a link to Bootstrap's content delivery network (CDN). Under the title of the document, add this code to the link tag:

```
<link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.mi
      integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGg
      crossorigin="anonymous"
    />
```

Next, add a link to our stylesheet under the link to Bootstrap's CDN. Since we created a stylesheet and index.html file at the same time, we'll just link to the style.css file that's in our css folder.

```
<link rel="stylesheet" href="static/css/style.css">
```

Now we can begin setting up the body of the HTML, where we'll store our components. To get started, within the `<body />` tags, add a `<div />` with a class of `"wrapper."`

```
<body>
    <div class="wrapper">
```

```
    </div>
</body>
```

The wrapper class adds a bit of extra functionality to Bootstrap. It helps group the elements (e.g., title, paragraph, table, and filters) and specifies the styling in our stylesheet.

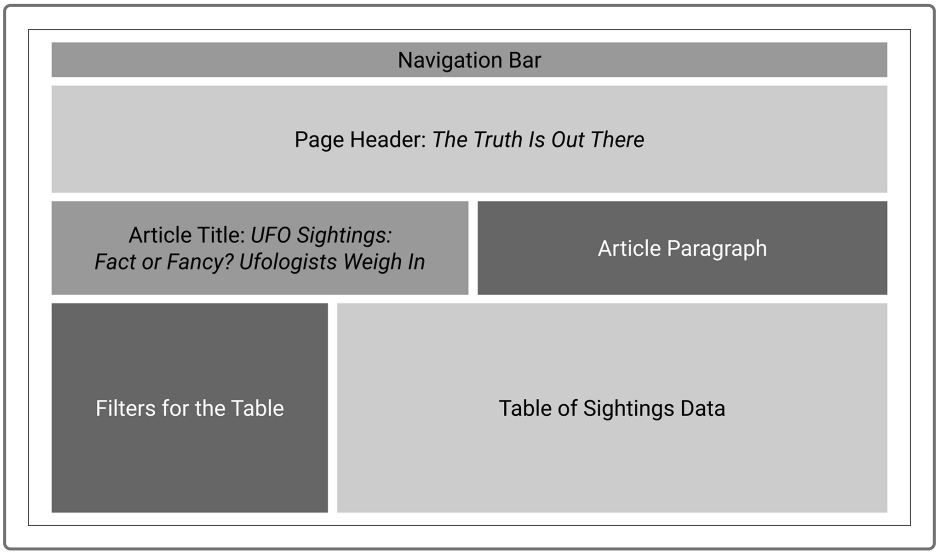
Now we can start adding the individual components.

The Bootstrap Grid System

Take a look at the storyboard we helped Dana create and consider how to employ the Bootstrap grid system.

REWIND

The Bootstrap grid system helps organize a webpage's content into containers, rows, and columns. A page can contain up to 12 columns per row and as many rows and containers as needed for content display. It also allows for a responsive webpage that will resize for the viewing area or screen size.



There are many different components to build. Let's start with the first one we see: the navigation bar (or navbar), where we'll add the functionality to reset our table filters.

Build the Navbar

Within the wrapper we created earlier, we'll add a new component: a nav tag with a class of `"navbar navbar-expand-lg."`

```
    <nav class="navbar navbar-expand-lg">

    </nav>
```

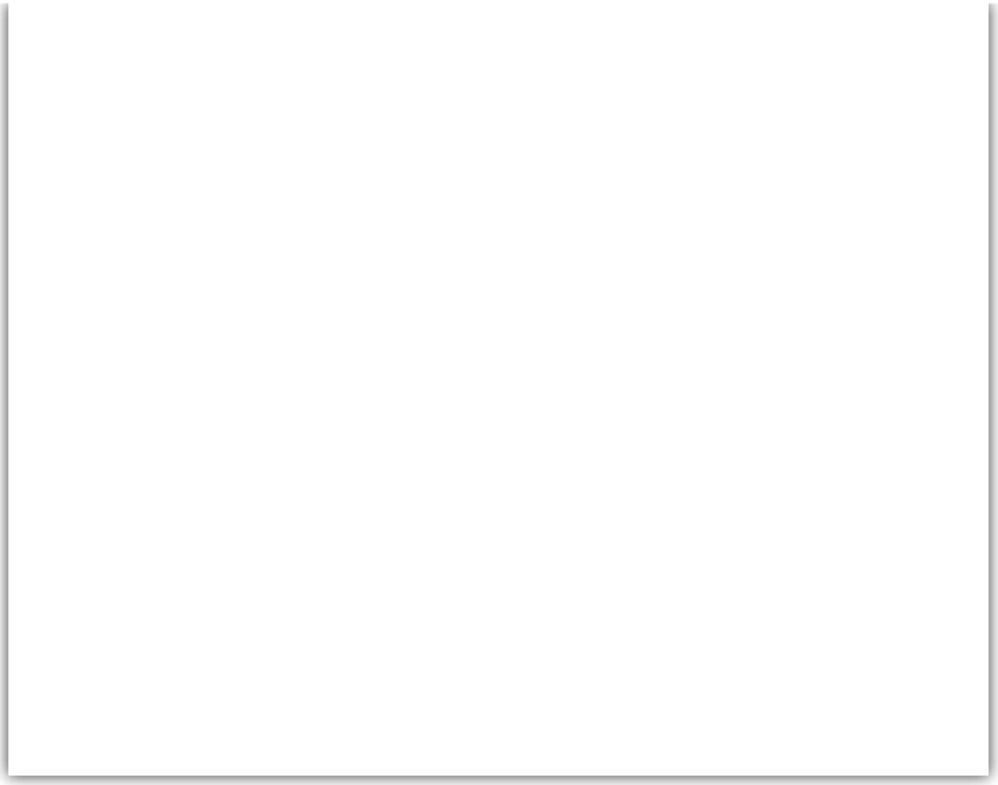
These classes are specific to Bootstrap's built-in styling; `"navbar"` indicates to Bootstrap that we want a component that fits across the top of the page. Specifying `"navbar-expand-lg"` provides extra responsiveness to the default navbar behavior. When the viewing area is reduced from a large to a smaller screen, the navbar will collapse or resize itself smoothly.

Now we need to add functionality to our navbar. In this case, we don't need to redirect readers to another section of the webpage. Instead, we want to reset the webpage after a filter has been applied to the table. This is accomplished by linking to the homepage, `index.html`.

To add a link, we'll nest an `<a />` tag within the `<nav />` tags.

```
    <nav class="navbar navbar-expand-lg">
      <a class="navbar-brand" href="index.html">UFO Sightings</a>
    </nav>
```

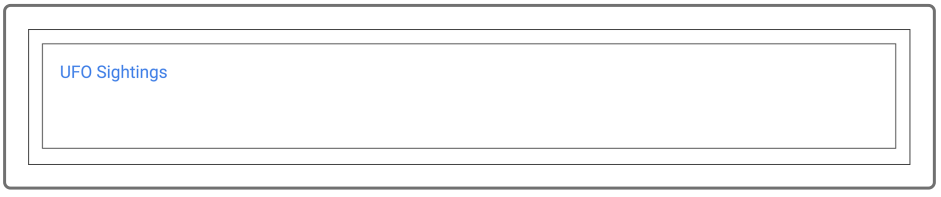
There are a few things happening within this new tag.



After adding `"navbar-brand"` to the tag (so there is less styling to worry about later), we also added an `href` that points to the `index.html` file we're working on. When a user clicks on that link, the page will reload and the default unfiltered table will appear, ready for new input.

We also need to display text in the navbar and complete the link. Dana has added "UFO Sightings" as the text for now.

Let's test to make sure our link to Bootstrap and the navbar is implemented correctly. Find your `index.html` file in your repo and open it with your default browser. The website should have a line of text that reads "UFO Sightings" in the top left corner.



The design is plain now, but we'll return later to customize.

Add the Jumbotron

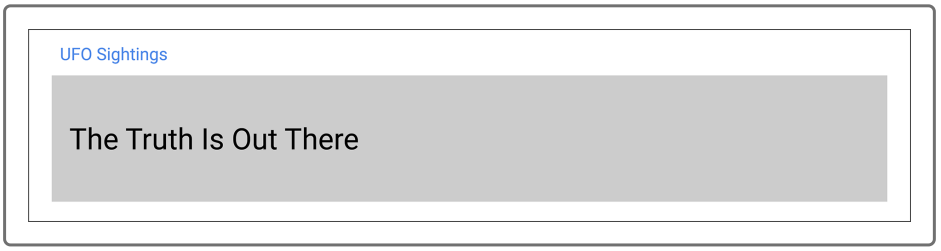
The first element, a navbar, has been added. Next, add the jumbotron. Because it's a new element and completely separate from the navbar, we want to add the new code beneath the existing code.

Let's set up the jumbotron by adding `div` tags, then the proper class.

Bootstrap looks for certain classes within HTML tags to indicate where to apply styling, such as by adding a `"jumbotron"` class to a `div` tag. Text nested within these tags will have visual enhancements automatically added. For example, a header tag nested within a jumbotron will be larger and bolder than a header tag on its own. In our code editor, let's add a jumbotron with a header that reads "The Truth Is Out There."

```
<div class="jumbotron">
  <h1>The Truth Is Out There</h1>
</div>
```

After saving this file, refresh the page we have open in our browser.



It's still rather plain, but we'll spruce it up after we finish assembling the other components.

Add the Article Title and Paragraph

So far, the webpage has the navigation established and a header that pops (and even more so after we customize it). Now we'll start getting into the content that Dana wants to display. In this section, we'll add the article title and paragraph. According to our storyboard, we'll need to utilize the grid system, which will let us assign screen space to each element.

The grid system, consisting of containers, rows, and columns, will need to be set up in a certain order: first the container, then a row, followed by how many columns each element will require. Create the container and row first.

In your text editor, create a `div` with a class of `"container-fluid,"` then nest a row within it.

Adding `"container-fluid"` to the `div` will ensure that both elements we're adding will span the width of the viewport. The `"row"` class makes sure that the title and paragraph will align neatly along the page.

```
<div class="container-fluid">
    <div class="row">
    </div>
</div>
```

Now we can add our columns. According to our storyboard, the title requires less width than the paragraph.

Let's assign four columns to the article title and the remaining eight to the paragraph. Remember that each element will be within its own `div`.

```
        <div class="col-md-4">

        </div>
        <div class="col-md-8">

        </div>
```

Now that the scaffolding is in place to hold the title and article, we can insert the title and paragraph Dana has chosen. Using an `<h3 />` tag, nest the article title ("UFO Sightings—Are They for Real? Ufologists Weigh In") in the first column.

```
<h3>UFO Sightings: Fact or Fancy? <small>Ufologists Weigh In</small></h3>
```

NOTE

> The `<small>` tag we've nested in will add a little bit of extra styling out of the box, too. Adding it will help de-emphasize the second portion of the title.

In the second `div` we added, the one that uses eight columns, let's add Dana's article paragraph. Here is what needs to be added:

Are we alone in the universe? For millennia, humans have turned to the sky to answer this question. Now, thanks to research generously funded by W. Avy, a UFO-enthusiast and amateur ufologist, we can supplement our sky-searching with data analysis.

"The release of this analysis is well-timed: It coincides with the celebration of World UFO Day, which is a moment for ufologists around the world to connect, relax, and sample a range of UFO-themed snacks," said Dr. Ursula F. Olivier, the world's preeminent expert on circular sightings. "Citizen-scientists can be especially helpful in both cataloguing sightings—which is hugely helpful for us in our search for aliens—and in helping us celebrate the work that has already been done, such as this data visualization project, which will help us raise awareness of the ubiquity of sightings!"

Not everyone is ready to celebrate, however. Local CEO and vocal anti-alien activist V. Isualize reached out to our reporters to go on record as firmly opposed to any attempts to provide access to this data. "If there are aliens, they certainly would like to be left alone," she stated, before

directing us to the Leave Aliens Alone (LAA) community engagement initiative she founded and funds.

So what do YOU think? Are we alone in the universe? Are aliens trying to contact us, or do they want to be left alone? Dig through the data yourself, and let us know what you see.

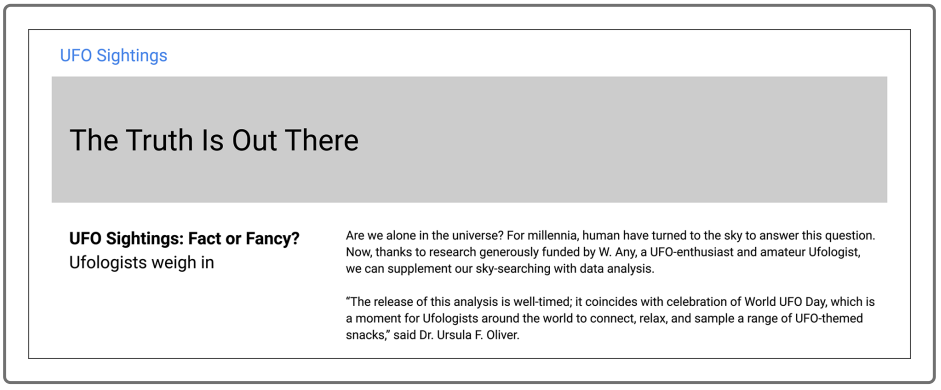
Our final code for this section should be similar to what's displayed below.

```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-4">
      <h3>UFO Sightings: Fact of Fancy? <small>Ufologists Weigh In</sm
    </div>
    <div class="col-md-8">
      <p>
        Are we alone in the universe? For millennia, humans have turne



      </p>
    </div>
  </div>
</div>
```

When we save this code and refresh our webpage, we'll see how the webpage is really starting to come together.



Create the Table Filter

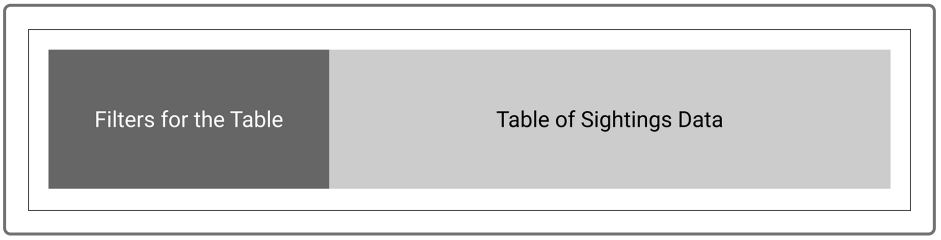
The next section of the webpage will tie together the JavaScript we've been helping Dana put together: We're going to build the section for the filter. The ids we created in our JavaScript code (`#filter-btn` and `#datetime`) will come into play here.

First, we need to set up the next row that will hold the filter section and the table.

SKILL DRILL

Create a new fluid container, and then nest a row inside it. This row will be where we store our filter and the data table.

When the new row has been created, we need to determine how many columns to designate for the filter section and how many for the table. Let's look at our storyboard again.



Compared to the previous row, the filters section looks like it will require less column space than the article title. Let's create a `div` with space for three columns and a nine-column `div` for the table.s

```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-3">

    </div>
    <div class="col-md-9">

    </div>
  </div>
</div>
```

Let's build the filter first. This is where we'll be accepting user input, so we need to use the HTML-specific field: a form.

First, we'll add a `form` tag, then build the form by nesting additional elements within it. Let's give the form a name so that users will know what it's for. After the `<form />` tag, add a `<p />` tag and the text `"Filter Search"` to that.

```
<form>
  <p>Filter Search</p>
</form>
```

With this, we know what this new component is, but now we need to add what Dana's readers will be searching for. Our JavaScript code has a filter setup to search by date, so we'll need to add an input box for a date. We'll also need the button that we referenced (`#filter-btn`) so that searches can be executed.

To add these items as cleanly as possible, we're going to create a set of list tags, nested within an unordered list tag. We'll also include some Bootstrap classes to keep things extra neat.

In your code editor, let's begin by starting the unordered list. We're going to give this a class of `"list-group."` Using this specific class lets Bootstrap apply predetermined styling to the list. We can spruce it up further if we want to after we're done.

```
<ul class="list-group">

</ul>
```

Next we need to add the list items: one for the input, one for the button. Each `<li />` tag will have a class of `"list-group-item."`

```
<li class="list-group-item">

</li>
<li class="list-group-item">

</li>
```

Now let's add the date input field in the first `<li />` tag. We'll add two new HTML components here: label and input. The label will be used as a prompt to encourage users to input a date. The input field is where users will complete the input.

In your code editor, add the label tag with the following modifications:

```
<label for="date">Enter Date</label>
```

This label represents a caption for the date item. The text `Enter Date` serves as the actual label.

On the next line, let's add the input.

```
<input type="text" placeholder="1/10/2010" id="datetime" />
```

There are three things to keep in mind when looking at this input:

1. The `type="text"` means that the code will look for text to be input.
2. The placeholder is an example of a date to search, so users know both the location and the format to use when inputting a date.
3. The `id="datetime"` is what our JavaScript code will look for when the button is clicked and the function is executed.

Now we need to add our button. In the next `<li />` we'll add a button tag with a few additional attributes: the id we defined in our JavaScript code (`#filter-btn`), a type, and a class. Let's add this as well.

```
<button id="filter-btn" type="button" class="btn btn-default">Filter Table</
```

When the button is clicked, the input from earlier is picked up by our JavaScript code and then applied to the filter.

`type="button"` tells the browser that, by default, the button does nothing. However, our custom JavaScript script will overwrite the default behavior—as if the button is waiting for instruction.

The `class="btn btn-default"` attribute adds some Bootstrap styling to the mix to help keep the element neat and tidy. Finally, we've named our button "`Filter Table`," by nesting it between the opening and closing button tags.

Prep the Table for Data

With the filter in place, we're now able to build the table. In our app.js script, we use D3 to select the "`tbody`" HTML tag. We're going to build that component and link the JavaScript and HTML files.

The table and filter components are both inside the same container, so we only need to construct the table HTML.

An HTML table has several nested layers:

1. `<table>`
    - `<thead>`
        - `<tr>`
            - `<th>`
    - `<tbody>`

Each tag present in an HTML table is used to either designate a section of the table, such as the `<thead>` tag, or it holds information that will be displayed. If one of the tags is missing or out of order, the entire table may not function correctly (or be assembled at all). Let's start constructing our table and adding information to it.

In our code editor, let's begin by typing out the nested tags.

```
        <div class="col-md-9">
          <table class="table table-striped">
            <thead>
```

```
            <tr>
              <th></th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>
```

We've included more Bootstrap styling by adding the classes `"table table-striped"` to the table tag. This will present a table that is slightly striped to give variation between the rows of data.

The `<thead>` tag will have a few nested tags within it. This is our table setup: All of the information displayed as headers will be added into this tag and its nested components.

The `<tr>` component signifies that everything nested within it will be displayed as a row of data. We will immediately see its use as we add each column header with the `<th>` tag.

Take another quick look at one of our data.js objects.

```
{
  datetime: "1/1/2010",
  city: "benton",
  state: "ar",
  country: "us",
  shape: "circle",
  durationMinutes: "5 mins.",
  comments: "4 bright green circles high in the sky going in circles then
},
```

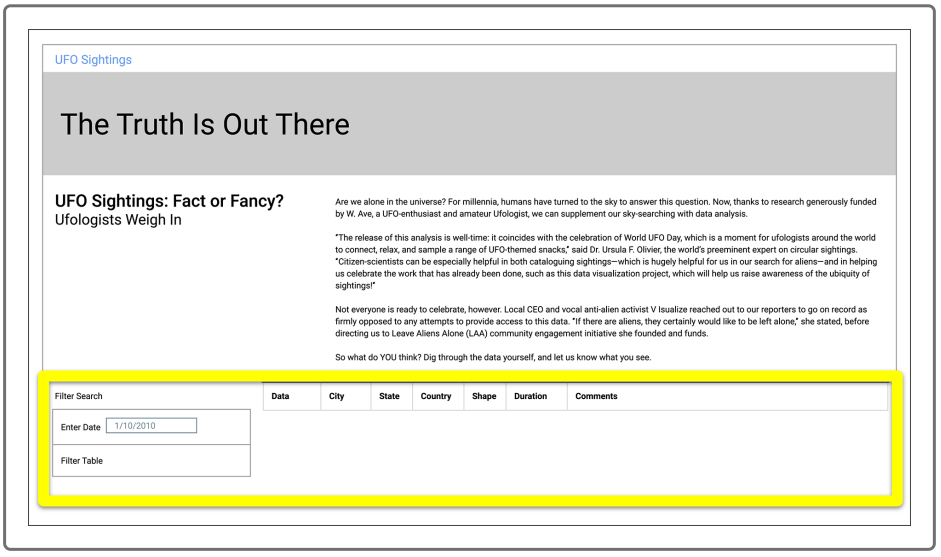
The information in our object is present as key-value pairs (KVPs). Each object will have the same key, but different values—these keys (such as `datetime`, `city`, and `state`) will be our table headers.

Back in our HTML code, we will need to add a `<th />` tag for each table header. Let's clean up the text a bit by using proper capitalization and spacing.

```
<th>Date</th>
<th>City</th>
<th>State</th>
<th>Country</th>
```

```
<th>Shape</th>
<th>Duration</th>
<th>Comments</th>
```

Save the file, then refresh the `index.html` page you have open in your browser—the only thing missing is the actual table data from the data.js file.



ADD/COMMIT/PUSH

Be sure to save your work and add, commit, and push it to your repo!