10.6.1 Customize the Appearance

The code you just created covers a lot of ground. Not only does it scrape new data on command, but it does it by incorporating your other code. By importing the script you wrote to scrape code, you were able to condense our new code into fewer lines, which is easier to read and troubleshoot.

You also added Flask, which allows you to put all of the data you've gathered into an easy-to-view web application. With each successfully completed step on the way to a full web app, Robin feels more confident about boosting her portfolio and eventually working for NASA.

The default Flask template isn't very interesting or inspiring, though, even with all of the neat data you've gathered. Robin really wants her web app to wow people, so to add a bit of extra polish, you'll use Bootstrap components to enhance the HTML and CSS of the `index.html` file.

Now that our code includes Flask, we need to create an HTML template for it.

Create HTML Template

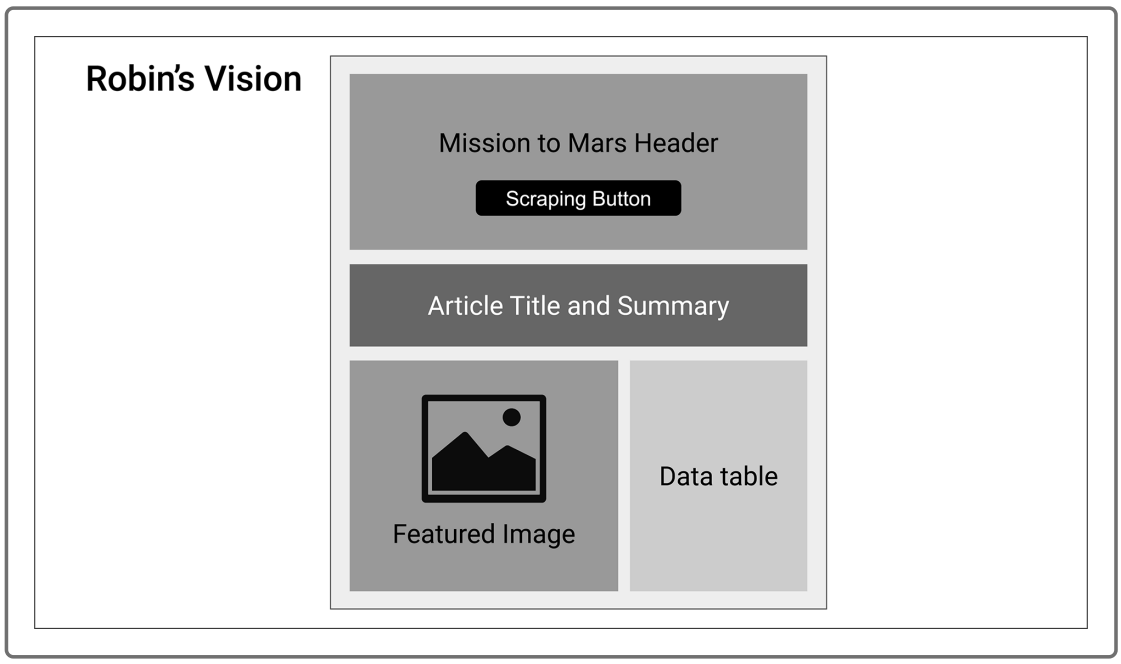
In our apps folder, create another subfolder named "templates"—this is where our HTML file will be saved. It's important to name and arrange these folders and files as we've lined out here. Flask templating works because Flask already "knows" what to look for (such as a "templates" folder and not a "Templates" folder); so, if either the capitalization or spelling is off, Flask won't recognize them. Make sure the capitalization, spelling, and location are all accurate, otherwise you'll find yourself in a debugging session.

Go ahead and create a file named `index.html` in this same templates directory while we're here.

SKILL DRILL

Open a new, blank `index.html` file with VSCode. Create a basic HTML template by using the exclamation point shortcut.

Let's think about how we want the page to look for a moment. We have three sections we want displayed on the web app: a news article title and summary, a featured image, and the table of Mars facts. We'll also want a nice header at the top to really help the app stand out—something like so:



Not only will we have to think about how we're laying these pieces out, but we'll also be adding Bootstrap in to really add some polish.

A benefit of Bootstrap is that it is responsive. By responsive, we mean webpages that react to different screen sizes. For example, webpages that resize images based on a screen size are considered responsive. Another example is when content is reorganized based on screen size: three items in a horizontal line on a large screen becomes three items stacked on top of each other when viewed on a small screen.

It's extremely popular and comes with loads of built-in components that make designing a webpage a much simpler process. Before we can do anything, we'll need to add Bootstrap's content delivery network (CDN) to our HTML.

A CDN is a group of servers that work together to provide users with content. In Bootstrap's case, it means that we use the Bootstrap data that lives on those servers and pick and choose what we need for our project.

An alternative to using a CDN is to download all of the components to our computers, which takes up space on our machines. Since we're only using a few of the components, it's more efficient to use the CDN.

IMPORTANT

> Like many tech tools and coding languages, Bootstrap has many different versions of release. We'll be using Bootstrap 3 to customize Robin's web app because it's a stable release.
>
> When developers designate a version of their product as "stable," they're saying that this version has been thoroughly tested and is as bug-free as possible. Newer releases are often getting kinks worked out, which can often lead to more research and time spent debugging your code.

In our `index.html` file, we need to add a `<link />` link for Bootstrap's stylesheet. The `<link />` is where we can access all of Bootstrap's components without needing to download the files themselves. We do this by referencing a CDN for Bootstrap in the `<link />`.

Before closing the `<head />` tag, add this link:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
```

Notice the URL in the href attribute? It's the CDN for the Bootstrap version we'll be using.

Adjust the `<title />` so it reads "Mission to Mars," then compare your index.html code to the one below. It should look similar, but revise it to match if needed.

```
<!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>Mission to Mars</title>
   <link
```

```
    rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min
  />
</head>
<body>
</body>
</html>
```

Now that we have the stylesheet selected, we can start adding Bootstrap components. Components include buttons, thumbnails, alerts—there are many. One of the biggest features that Bootstrap offers is its grid system, which is what makes Bootstrap so responsive and popular. But before we start putting components together and into a grid, we need to set up the container.

Set Up Containers

Each HTML page with Bootstrap follows the same layout: The very first tag is a `<div />` with a class of `"container"`. This is because a container is required for the grid system to work. In your HTML document, in the `<body />`, add the following:

```
<div class="container">
</div>
```

HTML can get unwieldy fairly quickly—especially when we continue to refine it by adding classes and ids—so keeping the code neat is important. Below is how the page should look now:

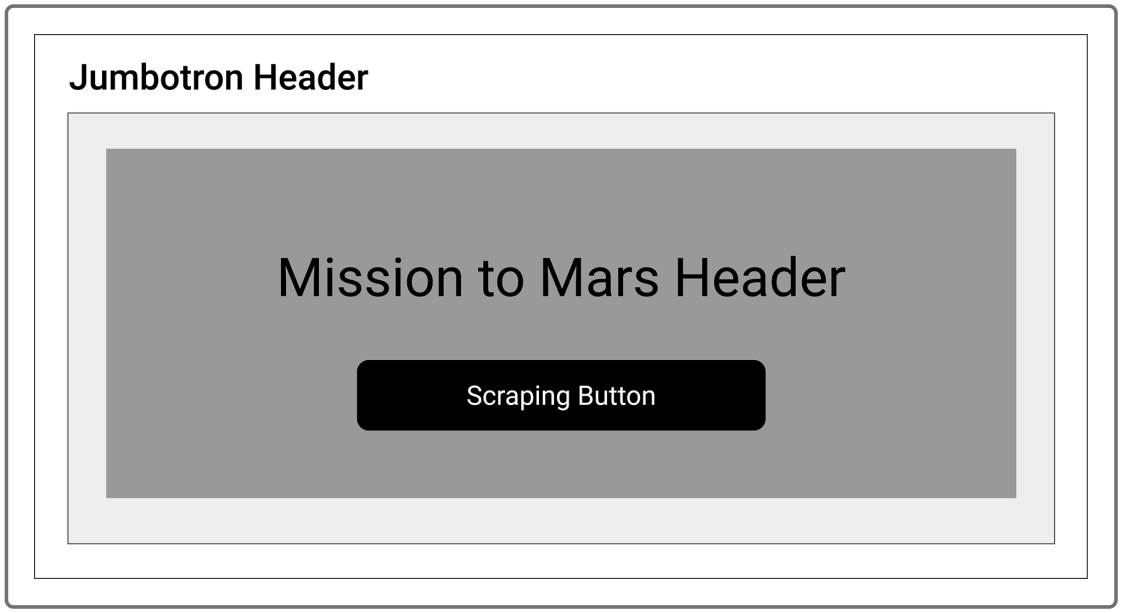
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <meta http-equiv="X-UA-Compatible" content="ie=edge" />
   <title>Mission to Mars</title>
   <link
     rel="stylesheet"
     href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min
   />
 </head>
 <body>
   <div class="container">
   </div>
</body>
</html>
```

Now that we have the basic layout for the grid system ready to go, we can add components. Let's start with the header.

Add the Header Using Jumbotron

For our header we'll use the Bootstrap Jumbotron component. In a non-programming context, a jumbotron is a very large video display screen used in venues like sports stadiums.

Similarly, Bootstrap's Jumbotron makes the header BIG. Here's where the Jumbotron header will appear.



This Jumbotron component enhances the older style `<h1 />` header without additional customizations through CSS (though we can certainly customize it further if we want to later). In the sample image above, we have our "Mission to Mars" title and Scraping button inside the Jumbotron. This way, both items are front and center on the web app.

We'll set it up by creating a set of nested `<div />` elements first. The first (outer) set has already been created. Next, create the inner `<div />` element inside it. Once they're in place, we'll add `class="jumbotron text-center"` to the opening tag to the inner `<div />` element.

By adding these specific class attributes, we're creating the Jumbotron component and centering the text that will be displayed through it. Now our code should look like the following:

```
<div class="container">
   <div class="jumbotron text-center">
   </div>
</div>
```

Now we've built our Jumbotron, let's add the main text. The next level of nesting, within the Jumbotron div, is where we'll add the main text and the

button we'll use to run our scraping script.

We already know that we want the highest-level header, so let's add an `<h1 />` tag. We also know that Robin wants to scrape data with the click of a button, so we'll add that too.

```
<div class="container">
   <div class="jumbotron text-center">
      <h1>Mission to Mars</h1>
      <p><a class="btn btn-primary btn-lg" href="/scrape"
                  role="button">Scrape New Data</a></p>
   </div>
</div>
```

The `<h1 />` tag is pretty straightforward. We don't need to dress it up at all because the Jumbotron is already doing that for us. The button is a little more complex, though.

First, we added a `<p />` tag to the Jumbotron, without any classes. This is because it's only there to help with formatting; it helps styling the content within the tag. Then we've added an `<a />` tag with a few different classes included.

The `"btn btn-primary btn-lg"` classes are also part of Bootstrap's button component (we're using Bootstrap to create a button instead of the `<button />` tag and element). There are three classes here because we're telling Bootstrap that we're using a button (`btn`), it's the primary color (`btn-primary`), and it's large (`btn-lg`).

An `<a />` tag is specifically used to link to other things, so we've also included the href (hypertext reference). The href is the link to another document or webpage. In our case, we're linking it to another component of our page, `"/scrape"`.

We've added a `role="button"` role so that the webpage knows this link functions as a button, not as a regular hyperlink. This just boils down to aesthetics, though.

The `role="button"` is required for screen readers, which are devices that assist people with vision impairments while they visit webpages. When a screen reader encounters role=\"button,\" it will know that it has found a button and not a simple anchor element (`<a />`).

Notice that the text in our `<a />` element is "Scrape New Data"? This is how the text will appear in our Bootstrap button.

NOTE

> Before moving on, double-check to ensure all elements and tags have been closed in the proper order.

Retake

Add the Mars News Article and Summary

Next we'll add in the news article and summary and use the grid system that Bootstrap offers.

Bootstrap Grid is a very useful tool because it provides extremely flexible and customizable layouts for all of your HTML components. It consists of a column and row layout with up to 12 columns per row. We're not required to use every column available, and we can mix and match how many items fit into a row.

It's also responsive, which means that it will adapt to whatever browser is being used to view content. Whether the app is viewed on a laptop, a desktop computer, or a tablet, the webpage content will adjust to best fit the screen size. We'll experiment with this as we continue to build the web app.

For the article and its summary, we'll want it to sit right under the header and span across the page. To span the entire page, we'll utilize all 12 of the grid columns. Let's set the grid up first.

Under the header, add a comment for clarity. This way we'll know where the code for each component is.

```
<!-- Mars News -->
```

Now we can add the code, which will be more complex than what we've written so far in HTML. There are a lot of nested tags, so we'll add one at a time and discuss them as we go.

First, create a new div with a class of `"row"` and an id of `"mars-news"`.

```
<div class="row" id="mars-news">
</div>
```

By adding a class of `"row"` we're telling Bootstrap that we're about to use the grid system and everything within this div will be horizontally aligned. The id `"mars-news"` helps us select it later if we want to customize it later with CSS.

Next we'll add a tag that specifies how many columns this component will be using. We want the article and summary to span the width of the page, so we'll use all columns. Update your code as follows:

```
<div class="row" id="mars-news">
    <div class="col-md-12">

    </div>
</div>
```

The `col-md-12` means that we want to use 12 columns scaled to a midsize desktop display.

SHOW HINT

Now we'll add the media portion: the article and summary. Because it's a news source, we want it to be displayed as media. There are two more tags we'll add to achieve this, so let's update our code with them.

```
<div class="row" id="mars-news">
   <div class="col-md-12">
    <div class="media">
        <div class="media-body">

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

With this update, we've nested in two more tags: one with a class of `"media"` and one with a class of `"media-body."` First, this tells Bootstrap that we want our content to be displayed as media, and then we insert the media within the body. Bootstrap already has the custom CSS set up for these classes. Now our setup is complete and we can add the article title and summary.

First, we'll add a header, this time a second-level one, so this section has a title. Nest the following `<h2 />` tag directly inside the media-body `<div />`.

```
<h2>Latest Mars News</h2>
```

Next, we need to add the name of the article we scraped. This is where Flask comes into play. We'll use a fourth-level header for the title of the article, placed beneath the `<h2 />` element we just created. We'll use double curly brackets to insert the title of the news article.

```
<h4>{{mars.news_title}}</h4>
```

The `mars.news_title` follows the syntax document.variable_name: In our scraping code, we saved the title of the article we scraped to the "`news_title`" variable and stored that in the `mars` Mongo document. When Flask encounters these curly brackets, it then looks to the string (in this case, a variable) stored within them and replaces the variable with the data it holds.

Since the title is also part of the section we're styling with media classes, we'll want it to also follow suit. By adding `class="media-body"` to the `<h4>` element, we're again telling Bootstrap to apply specific styling to the header. The final line of code will look like this:

```
<h4 class="media-heading">{{mars.news_title}}</h4>
```

Let's add the article summary the same way, only using paragraph tags. We're going to make use of Flask's ability to interpret the string within the double curly brackets again, this time to replace the variable with the paragraph text we scraped earlier.

```
<p>{{ mars.news_paragraph }}</p>
```

We didn't add any classes or ids to this paragraph tag because the default text is fine. If we want to change its appearance later, we can add our own CSS stylesheet.

All together with proper nesting, our code should look like this:

```
<!-- Mars News -->
<div class="row" id="mars-news">
     <div class="col-md-12">
       <div class="media">
         <div class="media-body">
           <h2>Latest Mars News</h2>
           <h4 class="media-heading">{{mars.news_title}}</h4>
           <p>{{mars.news_paragraph}}</p>
         </div>
       </div>
     </div>
   </div>
```

Add the Featured Image

Now we'll add the featured image that we scraped. This time, we want the image and the table of Mars facts to be next to each other, so we won't use the full 12 columns on the image. First, let's add a comment so we know which piece of data we're inserting.

```
<!-- Featured Image -->
```

Now we'll add the code that'll insert the image. We'll make use of an `<img />` tag, but instead of a string within the `src=` parameter, we'll use Flask's

ability to interpret strings (just like we did when we added the news article and paragraph).

Retake

Let's walk through these lines and how they've been assembled.

Each new item we're inserting now is on its own row, so the very first div will have a class of `"row."`

```
<div class="row" id="mars-featured-image">

</div>
```

We also added an id for later customization: `“mars-featured-image”`. This is so we can reference it using CSS if we want to fine-tune the appearance later.

Next, we'll add the div for the rows—eight this time. The number of columns must total 12, so we'll use the four remaining columns for the table data (which we'll add later).

```
<div class="row" id="mars-featured-image">
   <div class="col-md-8">

   </div>
</div>
```

Now we're going to add the code to insert the image. Add the following to your code:

```
<h2>Featured Mars Image</h2>
<img src="{{ mars.featured_image }}" class="img-responsive" alt="Responsive
```

First, we use an `<h2 />` element to add the title for this section: Featured Mars Image. The next line inserts an image using the `<img />` tag, but the tag alone won't actually insert the image. We also need to include the source, or link to where the image is. We do this by adding `src="{{ mars.featured_image }}"`—notice that we're once again using Flask's ability to interpret strings here.

When we add `class="img-responsive,"` we're using another built-in Bootstrap component that makes the image responsive. That means that the size of the image varies depending on the browser used, without us having to add extra code to do so.

The last portion, `alt="Responsive image,"` adds alt-text to our image. Alt-text is just text that will appear if the image doesn't load, or will be read by a screen reader if one is used. The benefit of alt-text lies in accessibility: visually impaired users will have the opportunity to better understand a webpage without actually viewing the image on it.

This final block of code should look similar to the one below:

```
    <div class="row" id="mars-featured-image">
      <div class="col-md-8">
        <h2>Featured Mars Image</h2>
        <img
          src="{{mars.featured_image }}"
          class="img-responsive"
          alt="Responsive image"
        />
      </div>
    </div>
```

Add the Mars Facts

We've gotten two of the three items we've scraped into our HTML. Let's add the third, our table of Mars facts.

Remember how the featured image only used eight of the 12 columns? This is where we'll use the remaining four. They're on the same row, though, so we need to be careful about where we place the new tags.

Retake

Let's take a look at the code we've created so far, starting with the featured image.

```
<!-- Mars Featured Image -->
<div class="row" id="mars-featured-image">
     <div class="col-md-8">
         <h2>Featured Mars Image</h2>
         <img src="{{ mars.featured_image }}" class="img-responsive" alt="Re
     </div>
</div>
```

When we add the table of facts, we want it to be on the same row as the featured image. This means that we want it to be nested inside the first level of `<div />` tags, but at the same level as the second set. In the code below, note the comments indicating the first and second level elements.

```
    <!-- Mars Featured Image →
    <!-- First level -->
    <div class="row" id="mars-featured-image">
      <!-- Second level -->
```

```
      <div class="col-md-8">
        <h2>Featured Mars Image</h2>
        <img src="{{mars.featured_image }}" class="img-responsive" alt="Resp
      </div>
    </div>
```

We'll add new code after the closing tags of the second `<div />`. In our editor, let's add a new column class, this time with four columns.

```
<!-- Mars Featured Image -->
<div class="row" id="mars-featured-image">
     <div class="col-md-8">
         <h2>Featured Mars Image</h2>
         <img src="{{ mars.featured_image }}" class="img-responsive" alt="Re
     </div>
     <!-- Mars Facts -->
     <div class="col-md-4">
     </div>
</div>
```

By adding the new set of div tags this way, we're completing the row and have used all 12 columns worth of space. Within these new tags, we'll start a new row and insert the table.

First, create a new div and add a class of "row." We'll also further customize it by adding a unique id: `“mars-facts”`.

```
      <div class="row" id="mars-facts">

      </div>
```

Second, we'll nest the header and table within the row. Using a fourth level header, we'll first add the "Mars Facts" title. Beneath that, we'll add the table.

Let's take a look at our entire page to see how it's coming together.

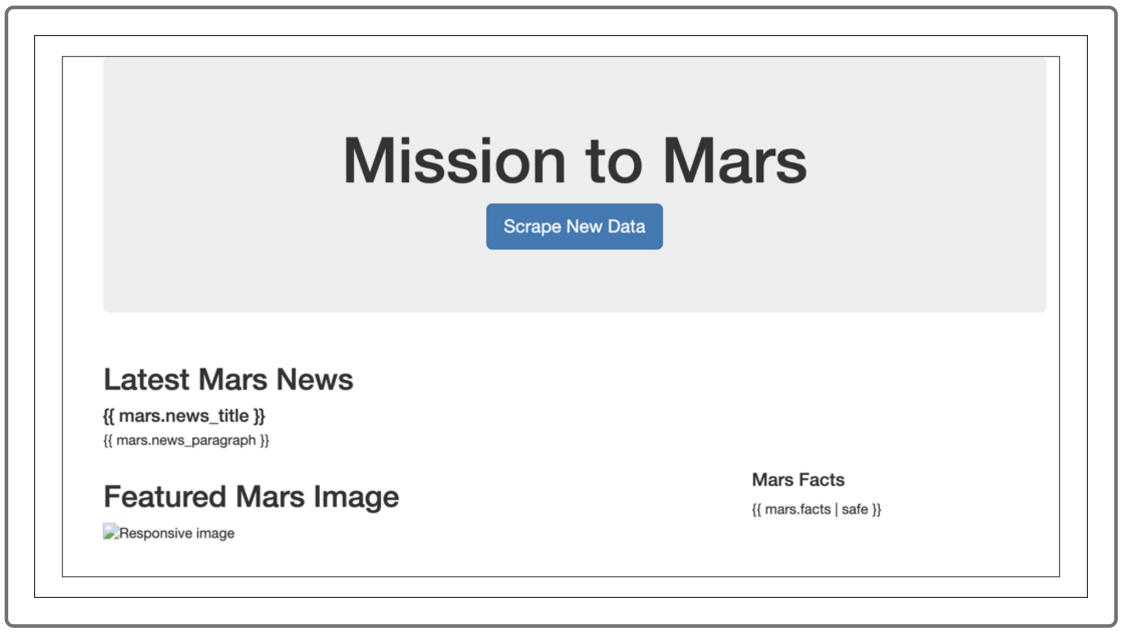
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Mission to Mars</title>
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.
    />
  </head>
  <body>
    <div class="container">
      <!-- Add Jumbotron to Header -->
      <div class="jumbotron text-center">
        <h1>Mission to Mars</h1>
        <!-- Add a button to activate scraping script -->
        <p><a class="btn btn-primary btn-lg" href="/scrape" role="button">
      </div>
      <!-- Add section for Mars News -->
```

IMPORTANT

> The line `{{ mars.facts | safe }}` accomplishes two tasks: it first references the facts table in the mars document, and it also tells the web browser that this code doesn't contain anything malicious. See the **official Template Designer Documentation (https://jinja.palletsprojects.com/en/3.0.x/templates/#working-with-automatic-escaping)** page for more information.

Adding only three items has created many, many lines of code. That's why indenting HTML properly and putting in descriptive comments with `<!-- and -->` is so important. Imagine if none of the tabs were aligned so neatly—it would be very difficult to see what's going on in there.

After saving the file, let's open it to see how it looks in a web browser. Without MongoDB running, it's just a shell—the variables we've put in place are still showing up as we wrote them initially. This is because Flask hasn't interpreted them yet; don't worry, they won't be that way for long!



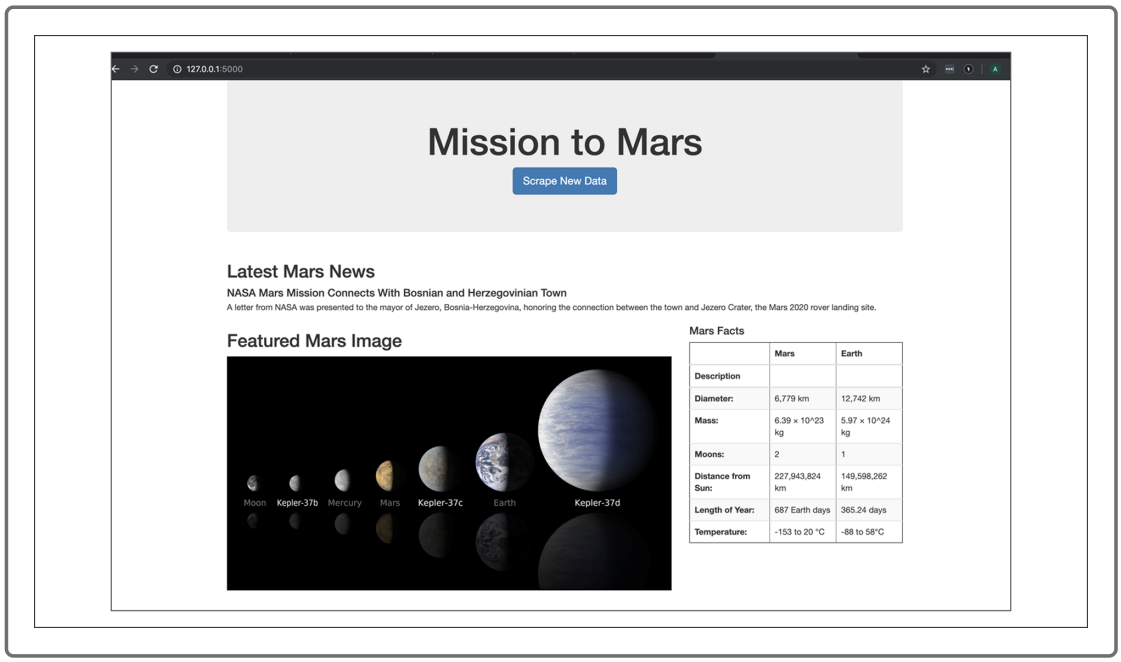
NOTE

> Your screen may not appear exactly as the one above, and that's okay!

But even a shell, we can see how it will all fit together like puzzle pieces cleanly clicking into place. After we click the Scrape New Data button, you'll see the terminal working and updating, and eventually a "Scraping successful!" message will appear on a blank page.

From here, we can navigate back to our localhost to see the updated webpage.

SHOW HINT



This is amazing! Everything we've been working on scraping has been collected into one location and presented in a clear and pleasing way. Bootstrap has made it easy to organize the data, too, without too much extra legwork on our part.