Going Mobile! - Using Bootstrap and Mobile Responsive Design

Objectives

- refactor existing website to use Bootstrap for layout

What is Bootstrap

Bootstrap is a framework that makes bringing HTML, CSS and Javasript to mobile devices in a super simple way. You won't have to spend a lot of time detecting whether your user is on a desktop, a tablet or a phone. Instead, you get to focus on creating great content for your users that will look great on all of their devices.

Adding Bootstrap to our web site

In order to begin using Bootstrap on our web page, we need to include a reference to the libraries. There are two pieces to the Bootstrap framework: Stylesheets (CSS) and JavaScript (js).

In the **index.html** page, we're going to add the two Bootstrap files to our <head> tag like this:

```
<head>
    <title>My First Web Page</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.c
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js
    <link href="app.css" rel="stylesheet" />
</head>
```

We could have downloaded these to our machine, but we are going to take advantage of the content delivery network for the bootstrap files. This takes the load off of our web server and potentially serves the file to the user faster.

At this point nothing has changed on our page. Adding the Bootstrap files will give us access to a collection of CSS classes and styles that will define how our page gets laid out on different devices.

Adding Containers

Bootstrap is based on having container elements that wrap all of the site contents. These containers then are subdivided into rows that each have 12 columns. We will then use classes to mark our code up for how many columns each set of content will take.

Let's start with the containers. We are going to use the "container-fluid" class to make a full width container that will resize based on the size of the browser viewport. This is the width of the screen and will be fairly small on a phone, but much larger on a tablet or desktop computer.

Let's start by adding this class to our page <header> in the **index.html** file.

```
<header class="container-fluid">
    <h1>Visit Anytown, USA!</h1>
</header>
```

Next, add another that will surround the main part of our web page including the both <aside> and the <section> tags that contain the bulk of the page content. Be sure to include all of the content except the <header> or <footer>.

```
<div class="container-fluid">
    <aside>...</aside>
    <section>...</section>
    <aside>...</aside>
</div>
```

Finally, add the "container-fluid" class to the footer of your page.

```
<footer class="container-fluid">
    Copyright 2015
</footer>
```

Save and view your web page in a browser. Resize the width of your browser and see how the header and footer resize as well as the content in the page.

Let's Add Some Navigation

Just under our header, we are going to use bootstrap to add a navigation menu to our page. We'll keep it simple, but Bootstrap will take care of everthing for us including the highlights, current menu selection and layout.

```
<header class="container-fluid">
    <h1>Visit Anytown, USA!</h1>
</header>
<nav class="container-fluid">
    <ul class="nav navbar-inverse navbar-nav">
        <li class="active"><a href="#">Home</a></li>
        <li><a href="http://www.bing.com/search?q=amazing%20places">Amazing Places<
        <li><a href="#contact">Contact</a></li>
    </ul>
```

```
</nav>
```

Before we take a look at the navigation we just added, let's add just a little styling to it so it will blend in with the styles of our page. In the **app.css** file, let's add the following styles:

```
nav {
    background-color: #222;
}

section, article {
    padding: 10px;
    background-color: #eee;
}

img {
    height: 100%;
    width: 100%;
}
```

Save and load the **index.html** file in your browser and notice the new navigation menu. As you resize your browser to a narrow window, you will see that the menu items go from being on the same line to being on their own lines. Thanks to Bootstrap, we don't have to spend a lot of time building this custom menu functionality.

We will add the content for the "Contact" link a little later in this module. Let's get back to Bootstrap layouts.

Now for the Rows

Next, we want to move on the main content of our web page. Once we have our containers, we can create a row inside of that container. That row will be the basis for the twelve columns that Bootstrap will use for layout. Any time we are working with a row, we want to keep in mind that the number of columns should always add up to twelve.

In our example web page, we want to have the first list take up three columns, the main content to take the next six columns and finally our second list to fill the final three columns.

In the **index.html** file, just after the surrounding our main content, add another and give it a class of "row".

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

We are now ready to subdivide the row with our content into the columns. Change both of the <aside> tags to have a class of "col-sm-3". This means on all devices small and up, create a space that is three grid columns wide.

```
<aside class="col-sm-3">
```

Since we did that for both of the lists, that will account for six of the columns in our grid. Now we need to change the main section of our content to have a class of "col-sm-6" to account for the other six columns.

```
<section class="col-sm-6">
```

Save and view **index.html** in a browser. Notice that the two lists we created now float to the right and left of our main content. What happens when you resize the window to a narrow screen similar to a phone? The our content automatically reflows as necessary.

More rows

Now that the content flows a little better when we resize the browser, let's take a closer look at our main content. We're going to make the content of the articles flow a little nicer with devices as well.

In the **index.html** file, lets add a with a class of "row" to the main section.

```
<section class="col-sm-6">
    <div class="row">
        <h2>Amazing Museums</h2>

    ...

    </div>
</section>
```

This will give us another row with another 12 column layout that we can begin to place each article into. Let's make the image take up three columns and the paragraphs fit in the other nine columns. Repeat this for each of the articles on the page.

```
<article>
    <h2>Amazing Museums</h2>
    <div class="col-sm-3">
        <img src="http://lorempixel.com/175/100/city/1" />
    </div>
    <div class="col-sm-9">
        <p>While most museums are a collection of old artifacts and stuff that no c
        <p>You won't find anything like these any where else in the world. Our muse
        <p>Come check them out...the fun is waiting for you.</p>
```

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

Save your work and load the **index.html** file in a browser. Try resizing the browser, and notice how the articles now flow nice and evenly on very narrow screens.

A Little Bit of Tug-o-war

When the content reflows, things tend to jump around. The question you have to ask, is "What is the most important content to be showing?" Currently, in a wide layout, our content is showing the main articles between our two lists. When we resize the browser window to a narrow view, the articles are still between our two lists...just further down the page that we would like. What if we wanted to show our articles first? We need to reorder our content to help us out here. Let's do that now.

Move the <section> tag and all of it's content above the first <aside> content.

```
<section class="col-sm-6">...</section>
<aside class="col-sm-3">...</aside>
<aside class="col-sm-3">...</aside>
```

Save and reload the **index.html** page in the narrow format. We got the look that we wanted. What about when we go back to the wide view? Things don't look the way we really want them now. The lists are both on the right side of the web page and it doesn't look very pleasing. Do we have to create two views of the data for this?

Bootstrap Push and Pull

Thankfully, Bootstrap has a few extra classes that allow us to flow the content to where it needs to go. We put the content in the order that we want it to flow, then we use the Bootstrap classes to tweak the details for the layout. Let's look at the Bootstrap "push" and "pull" classes. Push will move the content to the right. Pull will bring the content back to the left.

We will modify the main <section> class by telling it to push three columns to the right.

```
<section class="col-sm-6 col-sm-push-3">...</section>
```

Next, modify the first <aside> class by telling it to "pull" six columns to the left.

```
<aside class="col-sm-3 col-sm-pull-6">...</aside>
```

Save and reload the **index.html** page and look at the difference between the narrow width and the wide width browsers. Notice now the most important content is where it needs to be for each type of display.

About that Contact Info

Now that you have seen how to do column push and pull, let's add the content for our Contact information.

Because we added a link to our contact information, we will want to put in some HTML for that at the bottom of our page. Notice that putting the "name" attribute on the <a> tag it becomes a hyperlink that I can quickly forward a user to.

```
<div class="container-fluid">
    <div class="row">
        <div class="col-sm-6 col-sm-push-3"></div>
        <div class="col-sm-6 col-sm-pull-3">
            <a name="contact">Contact Us</a>
            <p>Anytown Chamber of Commerce</p>
            <p>123 Main St.</p>
            <p>Anytown, OH 43210</p>
        </div>
    </div>
</div>
```

Save your work and reload the page. Then use the "Contact" link in the navigation menu to scroll to the Contact information on the page. Be sure to check the behavior in both the full screen experience as well as a narrow small screen experience.

Push the changes to Azure

To close out this module, let's check in our changes and push them to Azure. You should already be set up for this from the previous modules, so if you haven't done the push to Azure before, you will need to back up and do that module first. From a "Git Shell" prompt, we will do the following steps:

1. Add all of the changed files to a checkin

   ```
   git add -A
   ```

2. Commit all of the changed files to your local repository

   ```
   git commit -m "Added bootstrap styling"
   ```

3. Push all of the code changes to your Azure repository

   ```
   git push azure master
   ```

Now, navigate to your website at http://yourwebsite.azurewebsites.net and confirm that your code has been deployed.