**Lab: Responsive Design**

<https://developers.google.com/web/ilt/pwa/lab-responsive-design>

**Overview**

This lab shows you how to style your content to make it responsive.

**What you will learn**

* How to style your app so that it works well in multiple form factors
* How to use Flexbox to easily organize your content into columns
* How to use media queries to reorganize your content based on screen size

**What you should know**

* Basic HTML and CSS

**What you will need**

* Computer with terminal/shell access
* Connection to the internet
* Text editor

**1. Get set up**

If you have not downloaded the repository and installed the [LTS version of Node.js](https://nodejs.org/en/), follow the instructions in [Setting up the labs](https://developers.google.com/web/ilt/pwa/setting-up-the-labs).

If you don't have a preferred local development server, install the Node.js http-server package:

npm install http-server -g

Navigate into the responsive-design-lab/app/ directory and start the server:

cd responsive-design-lab/app  
http-server -p 8080 -a localhost -c 0

You can terminate the server at any time with Ctrl-c.

Open your browser and navigate to localhost:8080/.

**Note:** [Unregister](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#unregister) any service workers and [clear all service worker caches](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#clearcache) for localhost so that they do not interfere with the lab. In Chrome DevTools, you can achieve this by clicking **Clear site data** from the **Clear storage** section of the **Application** tab.

If you have a text editor that lets you open a project, open the responsive-design-lab/app/ folder. This will make it easier to stay organized. Otherwise, open the folder in your computer's file system. The app/ folder is where you will be building the lab.

This folder contains:

* index.html is the main HTML page for our sample site/application
* modernizr-custom.js is a feature detection tool that simplifies testing for Flexbox support
* styles/main.css is the cascading style sheet for the sample site

**2. Test the page**

Return to the app in the browser. Try shrinking the window width to below 500px and notice that the content doesn't respond well.

Open developer tools and [enable responsive design or device mode](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#mobile) in your browser. This mode simulates the behavior of your app on a mobile device. Notice that the page is zoomed out to fit the fixed-width content on the screen. This is not a good experience because the content will likely be too small for most users, forcing them to zoom and pan.

**3. Set the visual viewport**

Replace TODO 3 in index.html with the following tag:

<meta name="viewport" content="width=device-width, initial-scale=1">

Save the file. Refresh the page in the browser and [check the page in device mode](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#mobile). Notice the page is no longer zoomed out and the scale of the content matches the scale on a desktop device. If the content behaves unexpectedly in the device emulator, toggle in and out of device mode to reset it.

**Note:** Device emulation gives you a close approximation as to how your site will look on a mobile device, but to get the full picture you should always test your site on real devices. You can learn more about debugging Android devices on [Chrome](https://developers.google.com/web/tools/chrome-devtools/remote-debugging) and [Firefox](https://developer.mozilla.org/en-US/docs/Tools/Remote_Debugging).

**Explanation**

A meta viewport tag gives the browser instructions on how to control the page's dimensions and scaling. The width property controls the size of the viewport. It can be set to a specific number of pixels (for example, width=500) or to the special value device-width, which is the width of the screen in CSS pixels at a scale of 100%. (There are corresponding height and device-height values, which can be useful for pages with elements that change size or position based on the viewport height.)

The initial-scale property controls the zoom level when the page is first loaded. Setting initial scale improves the experience, but the content still overflows past the edge of the screen. We'll fix this in the next step.

**For more information**

* [Set the viewport](https://developers.google.com/web/fundamentals/design-and-ux/responsive#set-the-viewport) - Responsive Web Design Basics
* [Using the viewport meta tag to control layout on mobile browsers](https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag) - MDN

**4. Use media queries**

Replace TODO 4 in styles/main.css with the following code:

@media screen and (max-width: 48rem) {  
  .container .col {  
    width: 95%;  
  }  
}

Save the file. Disable device mode in the browser and refresh the page. Try shrinking the window width. Notice that the content switches to a single column layout at the specified width. Re-enable device mode and observe that the content responds to fit the device width.

**Explanation**

To make sure that the text is readable we use a media query when the browser's width becomes 48rem (768 pixels at browser's default font size or 48 times the default font size in the user's browser). See [When to use Em vs Rem](https://webdesign.tutsplus.com/tutorials/comprehensive-guide-when-to-use-em-vs-rem--cms-23984) for a good explanation of why rem is a good choice for relative units. When the media query is triggered we change the layout from three columns to one column by changing the width of each of the three divs to fill the page.

**5. Using Flexbox**

The [Flexible Box Layout Module](https://www.w3.org/TR/css-flexbox-1/) (Flexbox) is a useful and easy-to-use tool for making your content responsive. [Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) lets us accomplish the same result as in the previous steps, but it takes care of any spacing calculations for us and provides a bunch of ready-to-use CSS properties for structuring content.

**5.1 Comment out existing rules in CSS**

Comment out all of the rules in styles/main.css by wrapping them in /\* and \*/. We will make these our fallback rules for when Flexbox is not supported in the [Flexbox as progressive enhancement](https://developers.google.com/web/ilt/pwa/lab-responsive-design#6) section.

**5.2 Add Flexbox layout**

Replace TODO 5.2 in styles/main.css with the following code:

.container {  
  display: -webkit-box;  /\* OLD - iOS 6-, Safari 3.1-6 \*/  
  display: -ms-flexbox;  /\* TWEENER - IE 10 \*/  
  display: flex;         /\* NEW, Spec - Firefox, Chrome, Opera \*/  
  background: #eee;  
  overflow: auto;  
}  
  
.container .col {  
  flex: 1;  
  padding: 1rem;  
}

Save the code and refresh index.html in your browser. Disable device mode in the browser and refresh the page. If you make your browser window narrower, the columns grow thinner until only one of them remains visible. We'll fix this with media queries in the next exercise.

**Explanation**

The first rule defines the container div as the flex container. This enables a flex context for all its direct children. We are mixing old and new syntax for including Flexbox to get broader support (see **For more information** for details).

The second rule uses the .col class to create our equal width flex children. Setting the first argument of the [flex](https://css-tricks.com/snippets/css/a-guide-to-flexbox/#article-header-id-13) property to 1 for all divs with class col divides the remaining space evenly between them. This is more convenient than calculating and setting the relative width ourselves.

**For more information**

* [A Complete Guide to Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) - CSS Tricks
* [CSS Flexible Box Layout Module Level 1](https://www.w3.org/TR/css-flexbox-1/) - W3C
* [What CSS to prefix?](http://shouldiprefix.com/#flexbox)
* [Using Flexbox](https://css-tricks.com/using-flexbox/) - CSS Tricks

**5.3 Optional: Set different relative widths**

Use the [nth-child pseudo-class](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child) to set the relative widths of the first two columns to 1 and the third to 1.5. You must use the flex property to set the relative widths for each column. For example, the selector for the first column would look like this:

.container .col:nth-child(1)

**5.4 Use media queries with Flexbox**

Replace TODO 5.4 in styles/main.css with the code below:

@media screen and (max-width: 48rem) {  
  .container {  
    display: -webkit-box;  
    display: -ms-flexbox;  
    display: flex;  
    flex-flow: column;  
  }  
}

Save the code and refresh index.html in your browser. Now if you shrink the browser width, the content reorganizes into one column.

**Explanation**

When the media query is triggered we change the layout from three-column to one-column by setting the [flex-flow](https://css-tricks.com/snippets/css/a-guide-to-flexbox/#article-header-id-5) property to column. This accomplishes the same result as the media query we added in step 5. [Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) provides lots of other properties like flex-flow that let you easily structure, re-order, and justify your content so that it responds well in any context.

**6. Using Flexbox as a progressive enhancement**

As Flexbox is a relatively new technology, we should include fallbacks in our CSS.

**6.1 Add Modernizr**

[Modernizr](https://modernizr.com/docs) is a feature detection tool that simplifies testing for Flexbox support.

Replace TODO 6.1 in index.html with the code to include the custom Modernizr build:

<script src="modernizr-custom.js"></script>

**Explanation**

We include a [Modernizr build](https://modernizr.com/download) at the top of index.html, which tests for Flexbox support. This runs the test on page-load and appends the class flexbox to the <html> element if the browser supports Flexbox. Otherwise, it appends a no-flexbox class to the <html> element. In the next section we add these classes to the CSS.

**Note:** If we were using the **flex-wrap** property of Flexbox, we would need to add a separate Modernizr detector just for this feature. Older versions of some browsers partially support Flexbox, and do not include this feature.

**6.2 Use Flexbox progressively**

Let's use the flexbox and no-flexbox classes in the CSS to provide fallback rules when Flexbox is not supported.

Now in styles/main.css, add .no-flexbox in front of each rule that we commented out:

.no-flexbox .container {  
  background: #eee;  
  overflow: auto;  
}  
  
.no-flexbox .container .col {  
    width: 27%;  
    padding: 30px 3.15% 0;  
    float: left;  
}  
  
@media screen and (max-width: 48rem) {  
  .no-flexbox .container .col {  
    width: 95%;  
  }  
}

In the same file, add .flexbox in front of the rest of the rules:

.flexbox .container {  
  display: -webkit-box;  
  display: -ms-flexbox;  
  display: flex;  
  background: #eee;  
  overflow: auto;  
}  
  
.flexbox .container .col {  
  flex: 1;  
  padding: 1rem;  
}  
  
@media screen and (max-width: 48rem) {  
    .flexbox .container {  
        display: -webkit-box;  
        display: -ms-flexbox;  
        display: flex;  
        flex-flow: column;  
    }  
}

Remember to add .flexbox to the rules for the individual columns if you completed the optional step 5.3.

Save the code and refresh index.html in the browser. The page should look the same as before, but now it works well in any browser on any device. If you have a [browser that doesn't support Flexbox](http://caniuse.com/#search=flexbox), you can test the fallback rules by opening index.html in that browser.

**For more information**

* [Migrating to Flexbox](https://www.sitepoint.com/migrating-flexbox-cutting-mustard/) - Cutting the Mustard
* [Modernizr Documentation](https://modernizr.com/docs)

**Congratulations!**

You have learned to style your content to make it responsive. Using media queries, you can change the layout of your content based on the window or screen size of the user's device.

**What we've covered**

* Setting the visual viewport
* Flexbox
* Media queries

**Resources**

**Learn more about the basics of responsive design**

* [Responsive Web Design Basics - Set the viewport](https://developers.google.com/web/fundamentals/design-and-ux/responsive#set-the-viewport)
* [A tale of two viewports](http://www.quirksmode.org/mobile/viewports2.html)

**Learn more about Flexbox as a progressive enhancement**

* [Progressive Enhancement: Start Using CSS Without Breaking Older Browsers](http://blog.formkeep.com/progressive-enhancement-start-using-css-without-breaking-older-browsers/)
* [Migrating to Flexbox by Cutting the Mustard](https://www.sitepoint.com/migrating-flexbox-cutting-mustard/)
* [Modernizr](https://modernizr.com/)

**Learn about libraries for responsive CSS**

* [Bootstrap](http://getbootstrap.com/)
* [Sass](http://sass-lang.com/)
* [Less](http://lesscss.org/)
* [Material Design](https://material.google.com/)

**Learn more about using media queries**

* [Using Media Queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries)