BUILDING A MOBILE FIRST WEB SITE

AIM

In this project, we are going to build a simple web page that should be flexible enough to use as a template for other pages. We are going to make the page mobile friendly as well as viewable in older browsers.

The files you need to use are on Blackboard.

- Download the zip file of resources.
- Unzip the HTML file *index.html*.
- Open *index.html* in your editor and in Google Chrome

VIEWING YOUR PAGES THROUGH HOMEPAGES.SHU.AC.UK

You can choose to view your pages locally or through the University's *homepages.shu.ac.uk* web server.

In order to do so save your files to the *f:/public_html*. If you create a folder called:

```
f:/public_html/mobileFirst
```

you will be able to view this through:

```
homepages.shu.ac.uk/~b12341234/mobileFirst
```

Where *b12341234* is your student number. Note the use of the tilde character (~).

You can also use the MD Assistance Centre (MDAC) to set up your *homepages.shu.ac.uk* space such that your pages can be viewed from outside the University.

MOBILE FIRST STYLESHEET

A requirement of this site is that it is mobile friendly. In order to do this, we will follow a technique known as 'mobile first'. This firstly involves designing a simple stylesheet for the site that will present the content in a mobile friendly fashion. The benefits of this are:

1. Focuses the designer on the key content.
2. The 'mobile' stylesheet will download first - other stylesheets for larger screens (laptops, desktops) will only be downloaded if required - as such this approach is kinder on mobile user's data allowance.

STYLING THE BODY AND CONTAINER

Create a file called *mobile.css* in the *css* folder. Add the following rules:

```
body {
  font-family: "Open Sans", helvetica, arial, sans-serif;
  margin: 0;
  padding: 0;
}
.container {
  width: 100%;
  min-width: 320px;
}
```

This sets the default font for our pages as well removing the 'native' margin and padding that belonged to the `<body>`. The class selector `.container` is used so that it can be used more than once through the file.

We'll also add some styling to the `.mainNav` to change the colour of our navigation bar:

```
.mainNav {
  background-color: #8d965b;
}
```

Attach this stylesheet to the *index.html* page by adding the following inside the `<head>`:

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

Test your page in Google Chrome using the developer tools to see how the page would appear on a mobile device.

STYLING THE NAVIGATION BAR

For mobile users, we would like a navigation bar to appear as a vertical list. As the HTML that forms the navigation bar is a HTML list `<ul>`, this will require only a limited about of styling. Firstly, to remove the bullet points, margin and the default left-hand padding add:

```
.mainNav ul {
```

```
  list-style: none;
  padding-left: 0;
  margin: 0;
}
```

To add a border to the bottom of each list item target the `<li>` as follows:

```
.mainNav ul li {
  border-bottom: 1px solid #ccc;
  padding: 5px;
}
```

The links will still have their default link styling, so we need a more specific CSS rule to target the `<a>` elements.

```
.mainNav ul li a {
  display: inline-block;
  width: 100%;
  font-weight: bold;
  text-decoration: none;
  color: #fff;
  padding: 8px 0;
}
```

This uses the display CSS property to change the `<a>` from `inline` to `inline-block` elements. HTML elements with a `display` value of `inline` cannot be assigned a `width`. Thus, by setting a `display` value of `inline-block` the element can be assigned a `width`.

VIEWPORT

In Google Chrome ensure you are in mobile view to test the page. At this point it may not be displaying quite as you would expect. The text on the navigation bar appears too small. This is due to the 'viewport'.

Web browsers on mobile devices are designed to cope with all web pages whether they have been made mobile friendly or not. To facilitate this, the mobile web browser behaves as if it has a much larger screen that it has in reality. Mobile browsers render pages in a virtual window known as the 'viewport' - that is wider than the physical screen dimensions. However, if a page is designed to be mobile friendly, then the browser needs to be told not to use its default viewport but to use the one set by the designer. This is achieved through the addition of a meta tag to specifically control the viewport behaviour. Add the following to the `<head>` of the HTML file.

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

This tells the mobile browser that the viewport is the same as the width of the actual device not the virtual viewport.

Now save and test your pages in the mobile emulator in Google Chrome.

DYNAMIC IMAGES

The pages are now being distorted by the images used. These images have dimensions of 700px by 400px. The width of 700px is been respected by the viewport of the browser, as it tries to fit the image into the whole page. We can fix this by adding a CSS rule to resize our images.

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

This tells all images via the HTML selector `img` that they should have a width of 100%. The 100% value refers to the width the element should assume of its parent element. As such they now appear as 100% wide of the `.container` given that no other widths have been applied.

Save and test your file through the Browser emulator. It should be looking a lot better now.

To help you understand how this technique is working try adding the following rule:

```
/* Just an experiment */
.mainContent {
  width: 50%;
}
```

When viewed all the images that are children of the `.mainContent` element now take up 50% of the screen space. Once you are happy with this idea remove the above rule.

From a design point of view, we probably want a little margin around the main text and images of our page so let's use that rule for `.mainContent` as follows:

```
.mainContent {
  margin: 0 10px;
}
```

USING FLEXBOX TO CREATE TWO COLUMNS

Having the main text and image appear in one column works fine on the mobile device. However, with bigger mobile devices there is scope to have a two column design. To achieve we could make use of the CSS Flexible Box Model - most commonly known as 'flexbox'.

Flexbox allows developers to assign a 'flex container'. The child elements of the flex container can then 'flex' to fill whatever space is available. These child elements are known as flex items. Flexbox is ideal for mobile first design as the flex items in a flex container can be made to flex in a variety of ways.

To introduce flexbox review the HTML element `div.news` to which we will first apply it. This `<div>` contains three child elements each of `div.newsItem`.

To make element `div.news` a flexbox container, simply set the display `property` to the new value of `flex`.

```
.news {
  display: flex;
}
```

Preview the page in the mobile view in Google Chrome. The contents of `.news` are now on one 'row'. To force the child elements of `.newsItem` to wrap add the `flex-wrap` property.

```
.news {
  display: flex;
  flex-wrap: wrap;
}
```

The design now reverts to how it was prior to adding flexbox. We now have to add some properties to the child elements (the flex items) of the flex container - these have a common class value of `.newsItem`.

```
```

```
.newsItem {
  flex-basis: 50%;
}
```

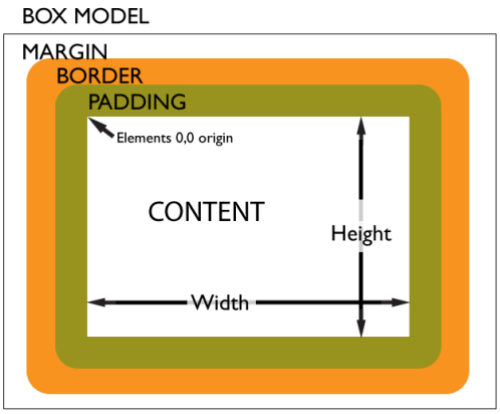
The above properties set a width of 50% to create the two columns.

BOX MODEL CALCULATIONS

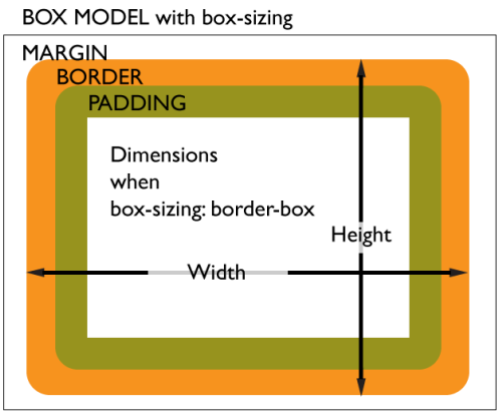
We now have two columns but the design is cramped. A solution might be to add some padding the `.newsItem` element with:

```
.newsItem {
  flex-basis: 50%;
  padding: 0 10px;
}
```

However, when tested this pushes the content back into one column. This is due to the Box Model that determines how values inside of the `width`, `height`, `margin`, `border` and `padding` are calculated by the browser.



As classic Box Model sees the `width` value inside of the `margin`, `border` and `padding`, we use the `box-sizing` property that changes the way width is applied.



As this way of calculating the box-model values is often more preferable to designer it makes sense to set this property for all the elements in our page using the global * star selector.

Add this rule to the top of your stylesheet.

```
* {
  box-sizing: border-box;
}
```

FORMATTING FOR LARGER SCREENS

The CSS added now gives us a working design for mobile devices. However, if you view the page in Google Chrome in the 'normal' desktop the current CSS gives a poor experience to desktop users.

To fix this add a second style sheet that will only be applied if the screen is greater than 720px.

As CSS rules are cascaded from one stylesheet to another the styling of the desktop view can be based on that of the 'mobile first' stylesheet. Therefore, the desktop stylesheet will amend or overwriting some of the properties already used, and in some cases, add some new ones. The technique for doing this is called 'media queries'.

WORKING WITH MEDIA QUERIES

Media queries allow the designer to ask questions about the browser and device been used to view the page. If the query matches then alternative stylesheets can be applied. For this project, we'll ask whether the screen has a minimum width of 720px. If so, we will add another stylesheet to the page. This is known as a breakpoint.

Add the following to the `<head>` of the *index.html* immediately after the link to the mobile.css stylesheet.

```
<link
  rel="stylesheet"
  href="css/desktop.css"
  media="only screen and (min-width : 720px)"
/>
```

This is essentially the same as a normal `<link>` element but with the addition of the `media` attribute. The `media` attribute contains the 'query'.

Why 720 pixels? This is a design decision based on the current state of mobile devices. It will mean that some larger screen devices (especially in landscape mode) will have this stylesheet applied.

The *desktop.css* has the following rule:

```
header,
.mainContent,
.row {
  max-width: 1200px;
  padding: 0 5px;
  margin: auto;
}
```

This limits our design to a maximum width of 1200px when viewed on devices larger than 720px.

THE NAVIGATION BAR FOR THE DESKTOP DESIGN

The *desktop.css* has some rules but more are required to create the desktop view.

First we'll add a little padding around the navigation. Add:

```
.mainNav {
  padding: 8px 0;
  height: auto;
}
```

We can use flexbox to have the navigation items flow horizontally.

```
.mainNav .row ul {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
}
```

We can also tidy up the borders on the links with:

```
.mainNav > div > ul > li {
  border: none;
  padding: 0;
}
```

USING FLEXBOX TO ADD A SIDE BAR

With the desktop design we now have plenty of screen space to place a side bar to the right-hand side of the page. In the mobile view let this content assume 'normal flow' and appear at the bottom of the content.

In the HTML locate the element `div.sideBar`. By collapsing elements down in the editor, you will be able to see that this element is a child of `div.mainContent`. Therefore, in *desktop.css* add a rule to main to make it flex container.

```
.mainContent {
  display: flex;
}
```

The flex items created by this rule are now `div.sideBar` and `<main>`.

As such we can set widths on them using `flex-basis` as follows:

```
main {
  flex-basis: 70%;
}
```

```
.sideBar {
  background-color: #ddceaa;
  flex-basis: 30%;
}
```

We'll change the `background-color` of the sidebar for good measure.

This will enable you to see the side bar content clearly in the browser as it will a new background colour.

The side bar now appears to the right-hand side. The `flex-basis` property of the flex item is used to set the initial size of the flex item. It in effect represents a target for the flex item to achieve. Be aware that other factors such as content, the flex container and the content of other flex items, will also come into play when rendered the dimensions of the item. However, in this example the widths of 70% and 30% will be respected.

USING FLEXBOX TO CREATE THREE COLUMNS

We also have plenty of space to re-arrange our news items. Change the `flex-basis` on the flex items with the following rule:

```
.newsItem{
  flex-basis: 33%;
}
```

USING FLEXBOX TO CREATE TWO COLUMNS

We can use the same techniques to style the 'hero' image and text in the `div.strapline`.

```
.strapline {
  display: flex;
}
.straplinetxt,
.straplineImg {
  padding: 0 5px;
  flex-basis: 50%;
}
.straplineImg > img {
  width: 100%;
}
```

CREATING A BURGER ICON MENU

Currently our navigation appears by default. However, we want to hide it and add a 'burger' menu to allow the user to access it. This will take up much less screen space.

There are a number of ways that we can create a 'burger button' such as using an image, but we'll do it purely through CSS. To do so add the following HTML inside the `header` element and after the `div.logo`.

```
<div class="burgerMenu">
  <div class="bars">
    <div class="bar1"></div>
    <div class="bar2"></div>
    <div class="bar3"></div>
  </div>
</div>
```

The HTML includes some nested `<div>` tags to create the bars of the burger button. This will need styling. Add the following to the *mobile.css* stylesheet.

```
.bars {
  float: right;
  cursor: pointer;
}
.bar1,
.bar2,
.bar3 {
  width: 24px;
  height: 4px;
  margin: 6px 0;
  background-color: #000;
}
```

The burger icon button is redundant in the desktop design. As such in *desktop.css* hide it with:

```
.burgerMenu {
  display: none;
}
```

SHOWING / HIDING THE MENU

We would like the menu to be initially hidden and then use the burger menu icon to toggle the visibility of the main navigation. As such in the *mobile.css* file amend the `.mainNav` rule as follows.

```
.mainNav {
  background-color: #8d965b;
  display:none;
}
```

By setting the `display` to none the burger menu will be initially hidden.

The necessary Javascript is in the file *js/main.js*. We'll also be making use of a Javascript library called jQuery to animate the menu. To add the Javascript to *index.html* by adding the following before the closing `</body>` tag.

```
  <script src="js/jquery-3.4.1.min.js"></script>
  <script src="js/main.js"></script>
```

Save and test. The menu should now appear and disappear 'toggle' when the burger menu is clicked.

ADD A SEARCH BOX TO THE HEADER

Add a search form as the last child of the `<header>` with:

```
    <div class="search">
      <form action="">
        <input type="search" name="search" aria-label="Search Term" />
        <button type="submit" aria-label="Search">
          <i class="fa fa-search"></i>
        </button>
      </form>
    </div>
```

In the *mobile.css* format the search box with:

```
.search > form {
  display: flex;
```

```
}
```

Then the format the search text and submit with:

```
input[type="search"] {
  flex-basis: 80%;
  padding: 12px 5px;
  border: 1px solid #ccc;
  border-right: none;
  -webkit-appearance: none;
}
button[type="submit"] {
  flex-basis: 20%;
  background-color: #4caf50;
  border: none;
  color: #fff;
}
```

We also would like a search icon on the search button. We can do this via an icon font like Font Awesome.

We can make use of Font Awesome by attaching their stylesheet with:

```
  <link
    rel="stylesheet"
    href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
    integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
    crossorigin="anonymous"
  />
```

Notice, how the search bar uses the `<i>` to attach the search icon from Font Awesome:

```
    <button type="submit" aria-label="Search">
          <i class="fa fa-search"></i>
    </button>
```

STYLING THE HEADER WITH FLEXBOX

To style the header content, which consists of the logo, Login/Register links, the burger menu and the newly added search bar we can again use flexbox.

Firstly, return to the *mobile.css* file and make the `<header>` a flex container with wrap enabled with:

```
header {
  display: flex;
  flex-wrap: wrap;
}
```

The header has four child elements that are now flex items. We would like them to appear with the following layout:

LOGO	LOGIN LINKS	BURGER MENU
SEARCH BAR		

To do so we'll let the first three items, `.logo`, `.loginLinks` and `.burgerMenu` flex to fill out 100% of the width and have the `.search` fill out another 100%. As the flex container is set to flex-wrap this should mean we can achieve the desired layout.

First set a maximum width for the logo image with:

```
.logo img {
    max-width: 400px;
    padding: 5px;
  }
```

FLEX GROW AND SHRINK

The `.loginLinks` and `.burgerMenu` flex items should occupy a fixed with. This can be achieved by stopping them shrinking or growing with the flex shorthand property of:

```
  flex: 0;
```

This is shorthand for:

```
```

```
  flex-shrink: 0;
  flex-grow: 0;
```

By setting these values to `0` the flex item will not try to either shrink or grow based on the space available as calculated by flexbox.

In the *mobile.css* add:

```
.loginLinks {
  /*  Don't set flex-basis as wrap will overrule it just stop grow and shrink */
  flex: 0;
}
.burgerMenu {
  /*  Don't set flex-basis as wrap will overrule it just stop grow and shrink*/
  flex: 0;
  padding: 13px 11px 0 11px;
}
```

We do however, want the `.logo` to grow and shrink. As such use:

```
.logo {
  /*  Don't set flex-basis as wrap will overrule it just set to grow and shrink */
  flex: 1;
}
```

This allow the `.logo` to resize based on the space available. Notice we have not used `flex-basis` to set the width as this will be overridden by the `flex-wrap: wrap` setting of the flex container.

For the `.search` we can now add:

```
.search {
  flex-basis: 100%;
}
```

For the desktop view we can amend the CSS so that the layout would appear (remember the burger button is hidden in this view):

Logo	Login/Register
	Search

To do so in the *desktop.css* add a rule for the logo as follows:

```
.logo {
  padding: 5px 0px;
  flex-basis: 85%;
}
```

… and rules for the `.loginlinks` and `.search`:

```
.loginLinks {
  flex-basis: 15%;
}
.search {
  padding: 5px 0;
  flex-basis: 40%;
}
```

In *desktop.css* add the header rule:

```
header {
  justify-content: flex-end;
  padding: 0 0 0 5px;
}
```

The search bar will be justified towards the end of the flex-container.

Finally, the `header` can be reduced in size by absolutely positioning the `.logo img`. To do so add the following:

```
.logo {
  padding: 5px 0px;
  flex-basis: 85%;
  position: relative;
}
.logo img {
  position: absolute;
  top: 0;
  left: 0;
}
```

By setting the `.logo` to be `position:relative`, it means the child `img` element can be set as `position:absolute`. This takes the image out of the 'normal flow' of the document and allows for the use of properties such as `top`, `bottom`, `right` and `left`. The values used will be relative to the parent container, in this case the `.logo`. This allows the micro management of the placement of the logo and if you experiment with the `top` value you will see it is placed above the other content in the page.

FORMATTING THE LOGIN AND REGISTER BUTTONS

The two links for login and register can be stylized to appear as link buttons. The styles we use here can be used for both mobile and the desktop. As our approach is mobile first we need only add them into the *mobile.css* file.

```
.loginLinks > ul {
  list-style: none;
  padding-left: 0;
  display: flex;
  justify-content: flex-end;
}
.loginLinks > ul > li {
  border: 1px solid #000;
}
.loginLinks > ul > li:first-child {
  margin-right: 4px;
}
.loginLinks > ul > li > a {
  padding: 4px 2px;
  color: #000;
  text-decoration: none;
  display: inline-block;
  width: 100%;
}
```

Notice use of the CSS pseudo class to select the first link in the list.

STYLING THE FOOTER

To style the footer content, again use the *mobile.css* stylesheet. The styles will again be inherited into by the desktop view. Add:

```
footer {
  background-color: #011424;
  color: #fff;
  padding: 0 10px;
}
```

We would like to add a list of social media links to the footer and we can again make use of Font Awesome.

Add the following HTML to the footer:

```
<div class="footerSocialLinks">
  <ul>
    <li>
      <a href="" aria-label="Twitter"
        ><i class="fab fa-twitter"></i
      ></a>
    </li>
    <li>
      <a href="" aria-label="Facebook"
        ><i class="fab fa-facebook"></i
      ></a>
    </li>
    <li>
      <a href="" aria-label="Instagram"
        ><i class="fab fa-instagram"></i
      ></a>
    </li>
    <li>
      <a href="" aria-label="YouTube"
        ><i class="fab fa-youtube"></i
      ></a>
    </li>
  </ul>
</div>
```

Notice the use of the `<i>` tags to add the icons.

Try styling these with:

```
footer .row {
  display: flex;
  flex-direction: column;
}
footer .row address {
  padding: 15px 0 5px 5px;
}
.footerSocialLinks {
  flex-basis: 20%;
}
.footerSocialLinks ul {
  display: flex;
  list-style: none;
  padding-left: 5px;
}
.footerSocialLinks ul li {
  flex-basis: 40px;
}
.footerSocialLinks ul li a i {
  color: #fff;
  font-size: 1.4rem;
}
.footerLinks {
  display: flex;
  list-style: none;
  padding-left: 0;
  flex-wrap: wrap;
}
.footerLinks li {
  border-right: 1px solid #ccc;
}
.footerLinks li:last-child {
  border-right: none;
}
.footerLinks li a {
  font-size: 0.8rem;
  width: 100%;
```

```
  display: inline-block;
  padding: 2px 5px;
  text-decoration: none;
  color: #fff;
}
.footerLinks li a:hover {
  text-decoration: underline;
}
```

BUILDING A FORM

Edit the HTML page called *contact-us.html* and add a HTML form with the following elements:

```
<form>, <fieldset>, <legend>, <label>

<input> - types text, email, tel, number, radio, checkbox, search, submit

<textarea>, <select>, <option>
```

When adding form elements, it is important that they have a name attribute. This becomes essential later when the form values are submitted to be processed in some way - for example by a PHP script. The name attribute and the value entered by the user are referred to as a name / value pair.

For example, the following creates a simple text field with a name of surname.

```
<input type="text" name="surname" id="surname" />
```

If the user enters a value of bloggs then the name / value pair submitted is surname=bloggs.

Tip: See http://www.mustbebuilt.co.uk/moving-to-and-using-html5/fabulous-forms/ for a review of the core HTML form elements you should get familiar with.

ADDING A FORM

To add a simple form on the page use:

```
        <form action="">
```

```
    </form>
```

The action attribute is concerned with where the form data will be sent to be processed (more on this in other labs).

Ensure all the form elements you add appear within the `<form>`.

STRUCTURING FORM CONTENT

When adding form elements, it is useful to also use HTML `<label>` tags to help identify what the form element is for. This is particularly useful for helping web page accessibility.

```
<label for="surname">Surname</label>
<input type="text" name="surname" id="surname" />
```

The `for` attribute is used to associate the label with a form element with a matching `id` value.

From a styling point of view it also helps to give these two elements a common parent ie:

```
<div>
  <label for="surname">Surname</label>
  <input type="text" name="surname" id="surname" />
</div>
```

Each new form element can be structured in this fashion:

```
<div>
  <label for="surname">Surname</label>
  <input type="text" name="surname" id="surname" />
</div>
<div>
  <label for="email">Email</label>
  <input type="email" name="email" id="email" />
</div>
```

STYLING FORM ELEMENTS

Following the 'mobile first' approach create two additional stylesheets for the form on the *contact-us.html* page.

The first stylesheet will style the form for mobile and then we will produce a second stylesheet that will be applied to the desktop using media queries.

As such your styles for the form will be attached as follows:

```
<link rel="stylesheet" href="css/mobile.css" />
  <link
    rel="stylesheet"
    media="only screen and (min-width:720px)"
    href="css/desktop.css"
  />
  <link rel="stylesheet" href="css/mobileForm.css" />
  <link
    rel="stylesheet"
    media="only screen and (min-width:720px)"
    href="css/deskForm.css"
  />
```

With the limited space of the mobile is it common practise to place labels above form fields. As the `<label>` element is an inline element create a rule with `display:block` to force a new line.

Other design aspects to consider for the mobile stylesheet is the width of the form elements. Try to maximise this with `width:100%`.

Try styling a form with a range of different HTML form elements. For the desktop stylesheet try to align the labels to the left of the form elements.

ANIMATING THE BURGER MENU

A nice touch would be animate the burger menu. We can do this using CSS tranistions. Firstly, indicate the length of the transition and apply it to all three of the bars of the burger menu. To do so add to the rule for the `.bar1` through to `.bar3`:

```
.bar1,
.bar2,
.bar3 {
  width: 24px;
  height: 4px;
  margin: 6px 0;
  background-color: #000;
  transition: 0.4s;
}
```

To illustrate what can be done we'll first of all fade out the middle bar of the three.

```
.animateBurger .bar2 {
  opacity: 0;
}
```

The Javascript used to animate the sliding of the navigation bar is also dynamically adding a class of `.animateBurger` to each of the three burger bars. In the above instance `.bar2` when clicked has the `.animateBurger` class added with a opacity of 0. But as the `.bar2` has a transition of 0.4s it will 'tranisiton' to this new state over 0.4 seconds.

As well as fading we can rotate using the CSS3 transform property. Add the following:

```
/* Rotate first bar */
.animateBurger .bar1 {
  transform: rotate(-45deg) translate(-6px, 8px);
}

/* Fade out the second bar */
.animateBurger .bar2 {
  opacity: 0;
}

/* Rotate last bar */
.animateBurger .bar3 {
  transform: rotate(45deg) translate(-6px, -8px);
}
```

SUPPORT FOR OLDER BROWSERS

Our design has relied on flex-box model that isn't supported by all browsers. Therefore, we should provide some fall back for users with older browsers.

There are a range of techniques for delivering fall back options. With Internet Explorer 5 to 9 supported something called IE conditional comments. These are HTML comments that older versions of IE interpret such that style rule can be applied to 'bug fix' issues in older version of IE.

For example this IE conditional commend <!--[if lt IE 9]> could be used to add some styles that will only be applied if the browser version if it is less than IE9.

```
<!--[if lte IE 9]>
<link rel="stylesheet" type="text/css" href="css/oldie.css" />
<![endif]-->
```

The *oldie.css* stylesheet is only loaded by Internet Explorer versions IE9 and below.

Fortunately, these older browser is becoming more scarce and rather than detecting the browser it is more sensible to see if a feature is supported. Feature detection is something that can be done with the modernizr (https://modernizr.com). Use this site to build a JS file to detect is features are supported, then create CSS rules for the no-SUPPORT scenario.

IPAD FRIENDLY MEDIA QUERIES

The page now tests well in Google Chrome mobile view. However, it is always recommended that you also test your pages on as many real devices as possible. When tested on an iPad the navigation bar was found to be a little cramped with some navigation items spreading over two lines.

Media queries can be applied in the `<link>` as we have seen, but they can also be written inside the CSS files themselves. We'll use this technique to fine tune the design for iPads.

In the *desktop.css* file add:

```
@media (min-width: 721px) and (max-width: 1122px) {
 /* ipad specific rules here */
}
```

The `@media` represents the media query to apply, in this case if the browser has a minimum width of 721px and a maximum width of 1122px. This should help us target iPads as they have a screen resolution of 1024x768. Inside the curly braces of the media query rule, we can now add the rules specific to the media query ie:

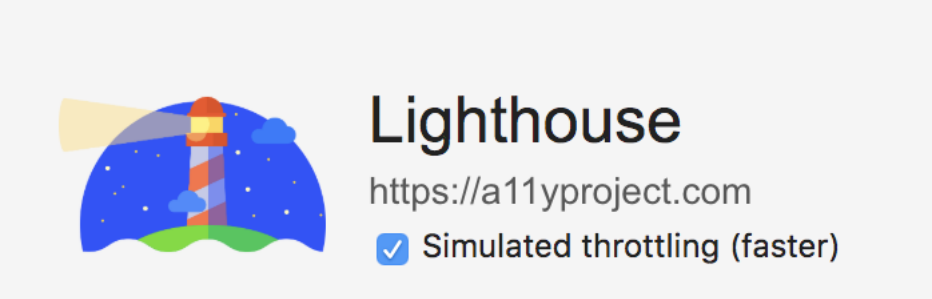
```
@media (min-width: 721px) and (max-width: 1122px) {
 /* ipad specific rules here */
 h1 {
  color: #f00;
 }
```

```
}
```

ACCESSIBILITY

Well-designed web pages should be accessibility to visitors who may have a range of visual and other impairments.

We can test how well our design performs by using The Lighthouse extension for Chrome.



This will allow us to run a report that will provide suggested improvements to the design.