# Making your site accessible

Step 1 – View your **interface** in the **browser** of your choice

Example: I would suggest opening up your interface in the **Chrome** browser

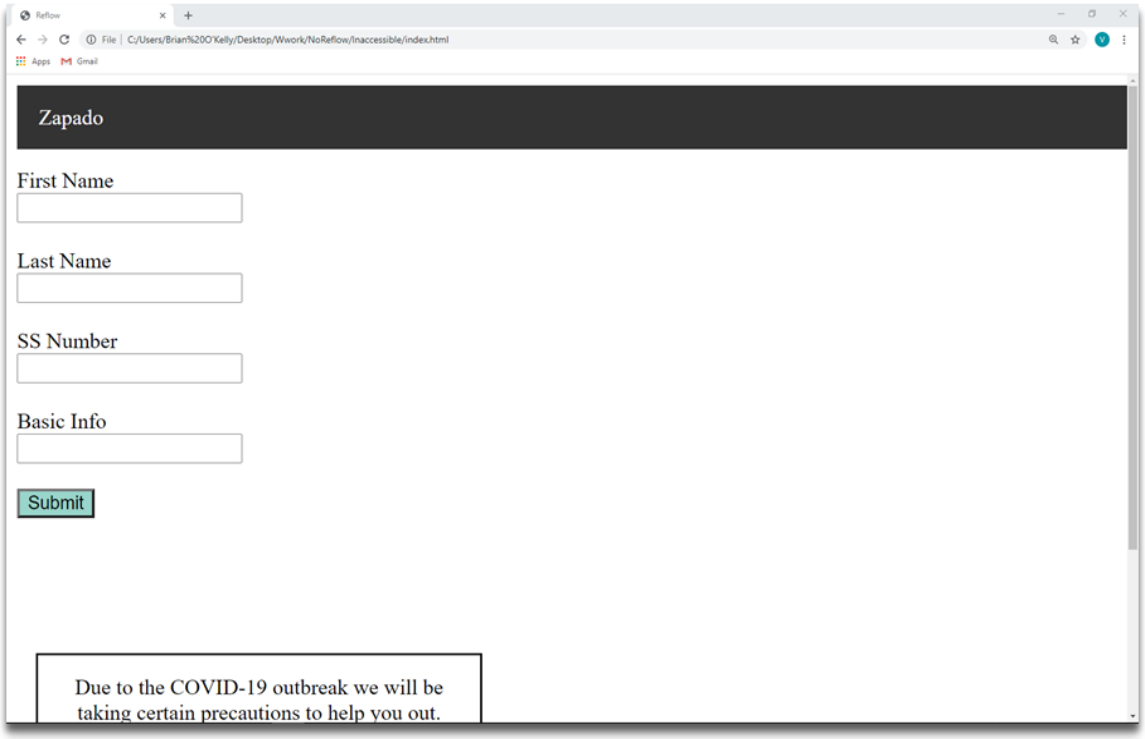


*Figure  SEQ Figure * ARABIC 1 - Open a browser of your choice*

Step 2 – Adjust the **viewport** to see how the **elements** move on the page

There are two commonly used methods for you to adjust your viewport represented by A (**manually adjusting the window**) or B (**using an inspector tool**)

Step A.2.1 - Select the **Restore Down** button in the top right corner of the page

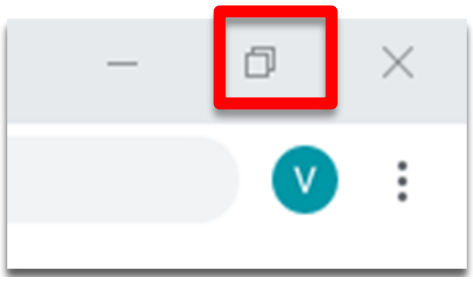

*Figure  SEQ Figure * ARABIC 2 - Select the Restore Down button*

Step A.2.2 – Now that you have access to sides of the window, you can use your mouse to resize the window via click and drag

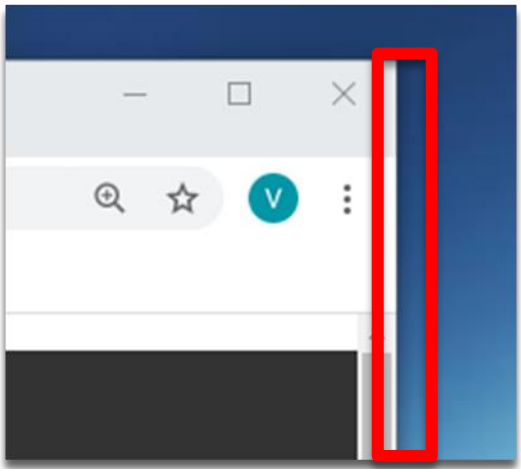

*Figure  SEQ Figure * ARABIC 3 - Select the side of the window and use your mouse to resize the window*

Step B.2.1 – Press the **F12** key or hold down "**ctrl + shift + i**" to active the **inspector tool**

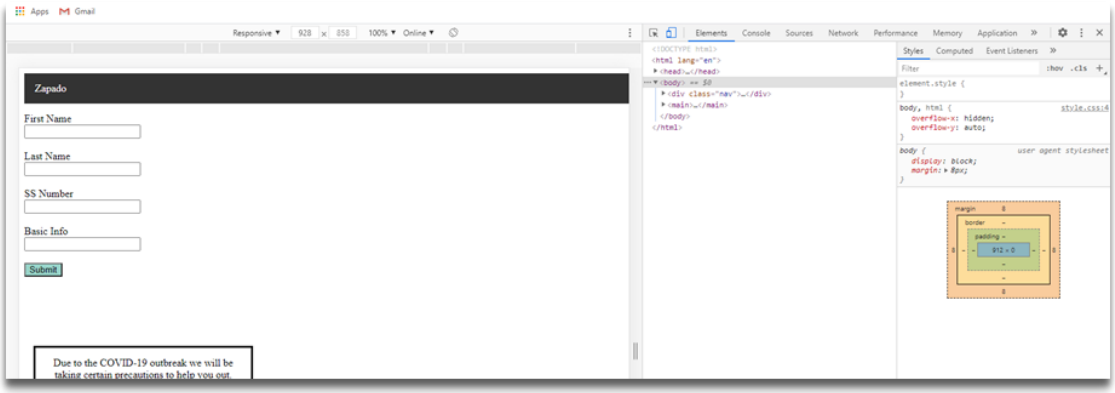


*Figure  SEQ Figure * ARABIC 4 - Opening the inspector tool*

Step B.2.2 – Use the inspector tool's **Device Tool Bar Selection Tool** in the top left of the inspector tool

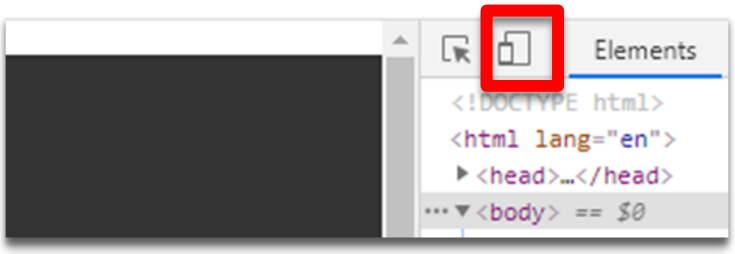


*Figure  SEQ Figure * ARABIC 5 - Selecting the Device Tool Bar Selection Tool*

Step B.2.3 – You chose a mode to change the **viewport** or just change it manually in the **Responsive** mode

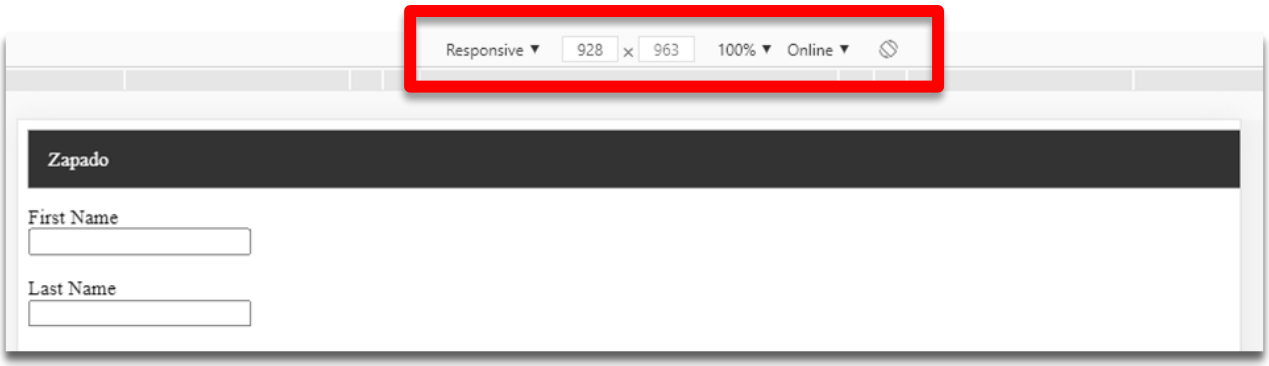


*Figure  SEQ Figure * ARABIC 6 - Changing the viewport size*

Step 3 – Now, you want to **adjust** the **viewport** to multiple different sizes to make sure that the content is still **accessible** to the users

In our example, it is hard to see elements when even when we are zoomed out. And our current inaccessible site does not have elements moving to accommodate the viewport (aka no responsive design).


*Figure  SEQ Figure * ARABIC 7 - Example page full size*

To fix this, we are going to create media queries to accommodate the row and column patterns we have to work with.

Step 4 -CSS Editing in style.css (Media Queries)

**Step 4.1 - Removing unnecessary CSS**

Lucky for us, despite having no-reflow accessibility, the interface does have a well-structured row and column structure. All we have to do is edit the CSS a bit.

First, let us remove all the styling that is much too specific. Starting with the first couple lines in style.css.

```
/*----------General----------*/
```

```
/*prevent accidental scrolling*/
body,
html {
  overflow-x: hidden;
  overflow-y: auto;
}
```

The script you see above prevents users form scrolling on the horizontal axis, which does have its purposes, but not here, so let us remove all of it.

Next, we will look at the **Form** section at line 34.

```
/*----------Form---------------*/
#form {
  position: absolute;
  width: 850px;
  margin-top: 60px;
  color: black;
}

#exForm {
  position: absolute;
  margin-top: 60px;
  margin-left: 1350px;
}
```

There is no need for absolute positioning as that will make reflow very difficult. With an absolute position, the ability to change based on viewport size is hampered.

Finally, we will go to the **Other Info** section which should now be located around line 34.

```
/*----------Other Info---------------*/
#info,
#exInfo {
  text-align: center;
  position: absolute;
  margin-top: 400px;
}
```

```css
#buttons {
  position: absolute;
  margin-top: 300px;
}

#submit {
  margin-left: 0px;
}

#exit {
  margin-left: 1350px;
}

p {
  width: 300px;
  border: solid black;
  padding: 14px;
  margin: 14px;
}

#info {
  margin-left: 0px;
}

#exInfo {
  margin-left: 1350px;
}

.hamburgerMenu {
  display: none;
}
```

There is no need for most of this, to make our lives easier it is best to just remove all of the above.

**Step 4.2-Adding CSS**

In our CSS file (style.css), likely around line 34, we want to add the following:

```css
/*----------Flex Box-----------*/
.row {
  display: flex;
}

.column {
  flex: 100%;
  padding: 10px;
  height: auto;
}
```

What we have just added is something called flexbox which allows our elements to resize in accordance with the viewport. This is a simplification of what flexbox is and what it can do. For more information outside the scope of our reflow example please check out more about flexbox.

For a visual aid, so we can more clearly make out where our elements are on the page, we will add the following:

```css
/*----------Other Info---------------*/
p {
  border: solid black;
  padding: 14px;
}

.submit {
  width: 20%;
}
```

The CSS addition seen above is intended on making it easier to identify the different elements of our interface when looking at it with the inspect tool.

Now, when we make the change the viewport to a mobile size, every element is fairly well-positioned, nice!

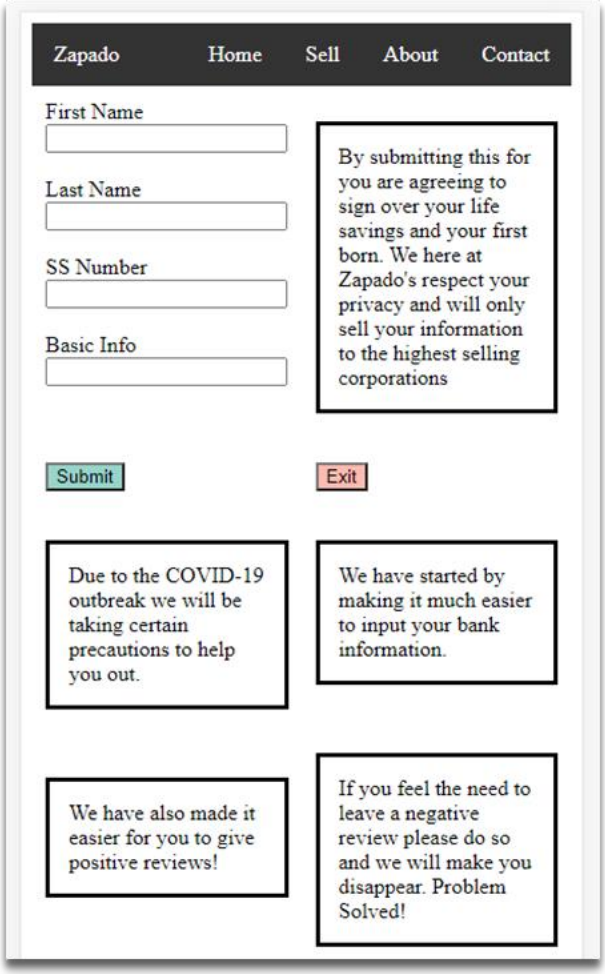


*Figure SEQ Figure * ARABIC 8 - Website in mobile view and accessible*

However, our goals are to explain media queries, so we will go one step further and use media queries.

**Step 4.3-Media Queries**

In the same style.css document around line 41, we will not be adding in media queries to fix reflow further.

Let's say that, for ease of viewing, we want only one column on the mobile view. In this example we can edit and add the following to our CSS:

```
/*----------Flex Box----------*/
.row {
```

```
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  width: 100%;
}

.column {
  display: flex;
  flex-direction: column;
  flex-basis: 100%;
  padding: 20px;
}

@media only screen and (min-width: 601px) {
  .column {
    flex: 2;
  }
}
```

Why would the media query be taking effect in the desktop view rather than the mobile view? Well, that is a little out of the scope of our **Reflow** guide, however, if you are interested and want to learn more please check out this guide to reflow.

Now we should have the ability to see a two columned layout on viewports bigger than 601px and single columned layout in viewport smaller than 601px.

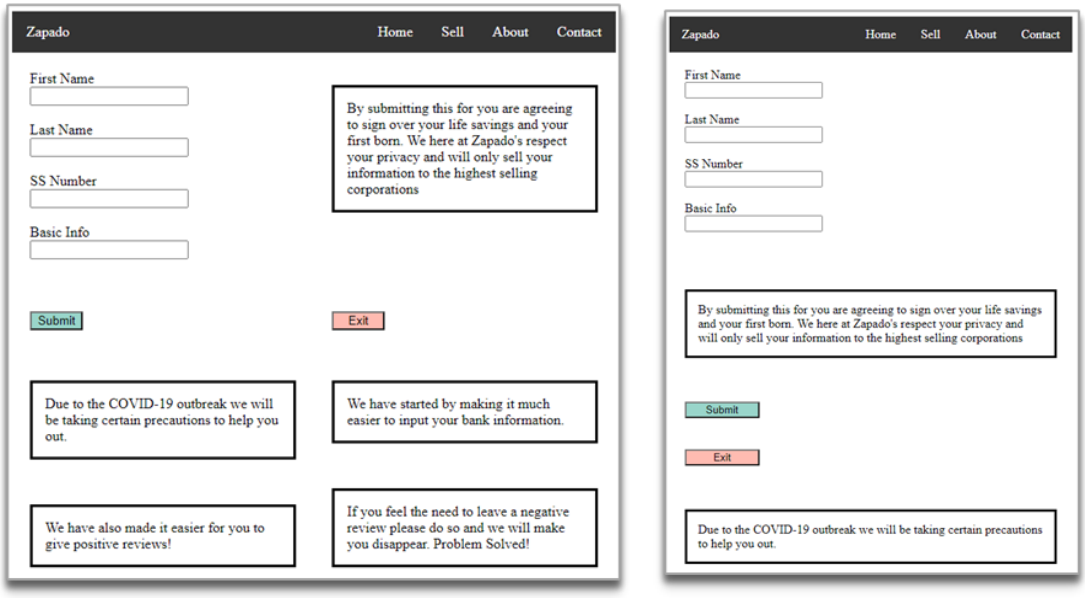


*Figure  SEQ Figure * ARABIC 9 - Side by side comparison of a viewport above and below 601px*

Step – 5 Navigation

While it is not necessarily a problem in our specific example, sometimes there are too many elements in the navigation causing either element to override each other or to take up too much of the user's attention. To avoid this bad interface design and possible cognitive trap, we should keep the content available, but shifted into a hamburger menu.

At the end of your style.css file please enter the following lines of script:

```
/*----------Hamburger Nav---------------*/
li.navLI {
  display: none;
}

li.hamburgerMenu {
  display: block;
}

@media only screen and (min-width: 601px) {
  li.navLI {
    display: block;
  }
  li.hamburgerMenu {
    display: none;
  }
}

i {
  color: white;
}

i:before {
  font-size: 28px;
  margin-right: 20px;
}
```

Here we are adding in the icon for the hamburger menu, setting its display property to none on larger viewports, and styling the icon.


*Figure  SEQ Figure * ARABIC 10 - Fixed site hamburger menu icon*

For the scope of this project, there is no functionality behind the hamburger icon. If you are interested in how to make it work, please check out this w3school's hamburger menu activity.