

ALX

WEBSITE

DEVELOPMENT

1. INDEX.HTML

HTML is the markup language with which all webpages are written. In this project, you will write a basic website about a topic of your choice.

Some pointers about HTML

You'll see HTML gives very basic outputs, as it typically only allows to stack vertically unmoving black texts, blue links, and images on a white background. To style your webpage differently, you will use another language later, called CSS (which you will discover in the next part of the project); and to make elements move and change dynamically, you will use again another language, called JavaScript (which you will discover in the part after next).

HTML is called a "markup language" as opposed to a "programming language", because it doesn't allow to program the computer and make it do everything you want, but only to mark text up with tags in order to structure it. The markup you write within the text tells the web browser (for instance,

Google Chrome, Mozilla Firefox, Internet Explorer, Safari and others) how to display the text to the user in the webpage.

Here are pieces of HTML code, and what a browser displays for them:

- ☐ `<p>This is a paragraph.</p>`
- ☐ This is a paragraph.

Note that `<p>` opens the paragraph (`<p>` it's an opening tag), and `</p>` closes it (it's a closing tag).

- ☐ `<p>This paragraph has a stronger part and an emphasized part.</p>`
- ☐ This paragraph has **a stronger part** and *an emphasized part*.

See how tags open and close within each other?

If you want to add a link on your webpage, you have to know that links are represented by `<a>` tags, like this: `<a>text to click on`. But when you want to create a link, you also need to tell what URL it links to when the user clicks. To tell the link what URL it should link to, you will need to add the URL in a `href` attribute of the opening `<a>` tag, like this:

- ☐ `<p>This is a paragraph with a`
`clickable link.</p>`
- ☐ This is a paragraph with a clickable link.

The value given to an attribute, just like for the `href` attribute here, is always put between the `"` characters, right after an `=` sign. The `<a>` tag accepts `href` as an attribute, because it needs to be told where to take the user when she clicks. Some other tags have attributes for their own purposes. For instance, image tags need to be told where the image file is, and they use the `src` attribute for that:

Notice that there is an opening `` tag, but there isn't a closing `` tag, because there's not especially any text to contain when you mean to insert an image. The `` tag is referred to as a "self-closing tag". In some versions of HTML, self-closing tags are also written like this: ``

- ☐ `<p>Here is an image: .</p>`



- ☐ Here is an image:

You can even mix things up a bit:

```
<p>  
  Here is a clickable image:  
  <a href="https://alx-apply.hbtn.io">  
      
  </a>  
</p>
```



Here is a clickable image:

(Notice that you can add new lines and blank spaces between tags and words in HTML, and it won't change the output as rendered by the browser.)

Before you get started on your project, you will probably like to know that a valid webpage contains at least this code:

```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>  
      This is the webpage's title, which will display in the browser's bar    </title>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
  
  </body>  
</html>
```

As you can see, all webpages always start with a `DOCTYPE` statement to express that the rest of the code is written in HTML.

Then, the HTML code itself is always entirely described between an opening `<html>` tag and a closing `</html>` tag. This tag always contains exactly two sub-tags: `<head>` and `<body>`:

- `<head>` contains some meta-information about the page, but not its content itself. For instance, it contains the title that the browser will display in its bar, and the "charset", which helps define which characters are used in the document (like: Chinese characters? Cyrillic characters?). "UTF-8" is the name of a charset that will do just fine for our needs.
- `<body>` which describes what your visitors can see. Your paragraphs, images, links, etc. must go between the opening `<body>` tag and the closing `</body>` tag. This is where you will write HTML code in the "HTML" part of this project.

Tips and links

- [You Only Need 10 HTML Tags](#)
- [HTML Cheat Sheet](#)
- [Official HTML Validator](#)

Your project

On your server, you have already found a file called `index.html` in your `/var/www/html` directory. As you've learned before, it is actually the HTML file that is being used when your browser renders the <http://3.236.220.208> URL.

This project will guide you to make your own website about a topic of your choice. If you haven't chosen yet what your website will be about, you should decide right now.

Make your first webpage valid

As you can see, the HTML code you have in `/var/www/html/index.html` does not match the minimum requirements of a HTML page, missing a few things like a `<title>` tag, the charset, etc.

Test the webpage with the [official HTML Validator](#) (choose to "validate by URI", and copy-paste your website's URL into the "address field"). You will see that it is not happy! First things first, fix the code in `index.html` so that it is valid. You will see that the validator then becomes much happier!

As you write more HTML code in your webpage, you should remember to go back to checking the validity of your webpage from time to time. It will prevent a lot of the kind of mistakes that you may miss otherwise.

Make a cool first webpage

In the `/var/www/html` directory, change `index.html`, so that it contains:

- at least 4 paragraphs;
- headings (titles) of level 1, 2 and 3, which are represented by `<h1>`, `<h2>` and `<h3>` tags;
- a clickable image of your choice (you can use the https://alxapply.hbtn.io/brand_alx/alx-logo.png image, or another one you find on the internet by using its full URL); and it may link to any URL online that you want.

Notice that if you save the `index.html` file, and refresh the <http://3.236.220.208> URL in your browser, the webpage will instantly reflect your changes.

Make a cool other webpage

Close the `index.html` file, and create another one, called `tweets.html`. You can either create it from scratch (and therefore, you will have to re-type all of the `<html>`, `<head>`, `<body>` tags, etc. that ought to be in all webpages), or you can create it by copying it from `index.html`, as you've previously learned to do, and replacing what you need to replace.

To see the webpage in your browser, you can access this URL: <http://3.236.220.208/tweets.html>. Note that the name of the `tweets.html` file should always be at the end of the URL like that, so that the browser knows which HTML file to render; `index.html` is an exception.

This page may contain anything you want, but it must:

- embed at least one tweet;

- contain a link to your index.html webpage, for users who may go back to your homepage; and your index.html should also contain a link to tweets.html, for users landing on your homepage, who want to see your tweets!

Here's the official documentation from Twitter about [embedding single tweets](#) in a webpage's HTML code.

Make a cool website

Decide on at least two other webpages and their content, that you could add to your website, and then create them as HTML files.

However, in order to start getting consistency throughout the website, all of the HTML files (including `index.html` and `tweets.html`) must now have the same structure:

- The `<body>` tag of all of your webpages (including `index.html` and `tweets.html`) must contain exactly three direct sub-tags in that order: `<header>`, `<main>` and `<footer>`.
- The `<header>` tag must contain an unordered list (the `` and `` tags) of links (the `<a>` tag) to each of your webpages. As you may have understood, this will serve as a navigation for your website.
- The `<footer>` tag must contain one paragraph with the sentence `Made by Alex haya for ALX`.
- The `<main>` tag must contain two direct sub-tags in that order: `<article>` and `<aside>`.
 - o `<article>` contains the content of your webpage: texts, links, images, tweets, ... This is the part of `index.html` and `tweets.html` that you have already written, and you must now write it as you like for the other pages.
 - o `<aside>` contains a single paragraph, just reading for now "placeholder to add comment thread later".

Once you're done making your basic HTML website, you can move forward to the CSS part of the project.

2. CSS

Now that you have a fully-fledged HTML website, you may notice that even though it is already using the same markup language as all webpages online, HTML doesn't offer many opportunities to make your website look the way you like, or even look good at all. To that end, we'll use the CSS language.

Some pointers about CSS

CSS is not a "programming language" either, as you can't program a computer to do everything you want with it, it is a "stylesheet language"; its only purpose is to apply style to HTML documents.

To write CSS code, you just write "CSS rules", each of which contain two things:

- First, what HTML tags you want to apply styling to (for instance: all links, all images, all paragraphs, etc.). This part of the CSS rule is called "the selector", because it allows you to select what HTML tags you want to style.
- Second, what styling you want to apply to them (for instance: have the text be a certain color, have it display in bold or italic characters, have the background be an image, ...). This part is called setting the CSS properties.

For instance:

```
a { color: red;
}
```

This piece of CSS selects all links in the page (the selector targets the `<a>` tags), and sets one property to them: they have to be red.

You can have several properties in the same selector. For instance

```
a { color: red; font-weight: bold;
}
```

Links will now be red and bold.

You can also have several selectors per rule, for instance:

```
a, h1 { color: red;
font-weight: bold;
}
```

All links and level-1 headings will be red and bold.

The list of existing CSS properties is pretty straightforward, but selectors can get pretty smart. For instance:

- if you want to be selecting only a few links of your choosing, you can add a "class" attribute to them (like this: ``), and select only those ones, using the `a.active_page` selector. This works with absolutely all existing HTML tags, and you can use any class name you like.
- if you want to select only the links pointed to webpages that were previously visited by your user, you can use the selector `a:visited`. `:visited` is called a pseudo-class. Another nice pseudo-class is `:hover`, for HTML tags that the user currently has her mouse on.

Tips and links

- [A list of all CSS properties](#)
- [A technical explanation about CSS Flexbox](#)

Your project

Some early styling

First, create an empty `styles.css` file in your `/var/www/html` directory. Then, in each of your HTML files, add those two lines within the `<head>` tag (do not confuse with the `<header>` tag!):

```
<link href="https://alx-apply.hbtn.io/level2/school.css" rel="stylesheet">

<link href="styles.css" rel="stylesheet">
```

If you refresh your webpages in your browser, they should now look a bit better! We just told your webpages to use the CSS rules described in those two files, and to apply them to your HTML code.

The `school.css` file is not needed on your server, as it is on another server, so that you can't modify it; this is the one containing the CSS rules that just changed your website so much. In the `styles.css` file, you will write any CSS rules you want to add; you can even rewrite rules that are already in `school.css`, as yours will override them.

Positioning

Even though each element of your webpages now has a different style, you may notice they still are stacked on top of each other, and that's probably not what you want. CSS brings a few different approaches to positioning that one may use depending on cases.

For this project, we're going to use CSS Flexbox, a recent set of CSS properties that work with recent browsers. You can ensure that it will work with your browser's version, by checking what your browser's version is, and that it is colored in green [in this table](#)

If it's not, you will have to switch to a CSS Flexbox compatible browser to complete this exercise.

Our goal is to get a layout that looks like this:



As a reminder, there are also two container tags playing critical roles here:

- `<main>` contains the area that includes `<article>` and `<aside>`.
- `<body>` contains all of them together.

Here are steps to get this layout, which you will have to write as CSS code in the `styles.css` file:

- Both container tags, `<body>` and `<main>` must be told that they are containers to flexible boxes: you need to apply the `display: flex` property to both of them.
- However, `<body>` contains a column of three boxes (`<header>`, `<main>` and `<footer>`), therefore you must apply the `flex-direction: column` property to `<body>`; whereas `<main>` contains a row of 2 boxes (`<article>` and `<aside>`), so you must apply the `flex-direction: row` property to `<main>`.
- Ensure the `<main>` tag keeps an automatic height and width, by applying the `flex: auto` property to it.
- To wrap up the layout, you want to be sure that your content (article) takes $\frac{2}{3}$ of the width of the page, and your aside takes $\frac{1}{3}$; you can assign to them the number of boxes they should fill in. This is done by applying the property `flex: 2` to `<article>` (using up 2 boxes), and `flex: 1` to `<aside>` (using up 1 box).
- Finally, you want to be sure that the user can scroll within your `<article>` and your `<aside>`. You can do this by applying the `overflow-y: auto` CSS property to both of them.

If you've done all of those properly, and your HTML structure is correct, you should get exactly the layout we were trying to get.

Do note that the exact rendering you're getting may be slightly different depending on your browser (namely, about whether header and footer are fixed or not when the user scrolls), but should always match the layout presented above.

Responsive web design

You may notice that the website keeps this layout when you make your browser window smaller, or visit it from a smartphone, and it's unpleasant to use that way for your users. No worries, we covered this for you: just add the attribute `class="works_on_smartphone"` on the `<body>` tag in your `index.html` file, and the layout you just created will degrade nicely as you resize the window!

But you'll notice, if you visit your website on a smartphone, that it will still have that layout, and just seem "zoomed out". That's because you need to adjust what is called the "viewport" of the browser when rendering the page. Hint: this gets done by adding a certain tag to your HTML code.

Some more styling

Your website is unique, and if you want it not to look like everyone else's, you may want to take some time to style everything of it as you wish!

What you're allowed to do:

- You may add any non-positioning-related CSS rules to styles.css (like colors, backgrounds, borders, ...)
- You may do whatever you want with the HTML content and the CSS that applies inside the `<article>` tag.
- You may add a logo to the top-left of your page. To keep it simple, rather than use an image, feel free to use a unicode character, [from this table](#) for instance. One way to make it work: add `<header>` a first item in the list in your , before all other list items, and just put the HTMLcode for the character in it (it starts with "&" and ends with ";"). To make it look like a logo, if you want the character to be bigger, you can add the `class="logo"` attribute on the `` tag, we added the CSS rule for you.

What you're not allowed to do:

- Do not change the layout strategy, done with CSS Flexbox, as described above. Feel free to experiment with positioning within the `<article>` tag if you wish; but please refrain to do so anywhere else.
- Do not change anything about the `<aside>` tag. It will be useful to you later in this project.

Once you like what you have, feel free to move on to the next project.

3. JAVASCRIPT

Your website probably looks very good now! But it also probably could use a little bit extra visual feedback to give the user...

Some pointers about JavaScript

This time, JavaScript is indeed a programming language! It can be used for many things, but in the context in which it runs in a browser (therefore, on a webpage), it typically changes the webpage and CSS rules in any way you like. JavaScript in the browser is event-driven, which means code typically executes when an event happens (like: the user clicks on something, or scrolls down the page, or resizes the window, etc.)

But let's experiment right away!

Your project: a smart thumbnail

The point of this part of your project is to display a large image as a smaller thumbnail, allowing the user to click on it to make it bigger.

During this part of the project, we're going to be writing code in two locations: in a .js file so that the webpages run it each time they load; and we'll also run code in the browser's JavaScript console, just to test stuff as the webpage is live in our browser. First, let's set both of those up.

Setting up the .js file

Create an empty file `behavior.js` in your `/var/www/html` folder; this is the file in which your JavaScript code loaded by your webpage will go. Then, so that the webpages know where to find it, add this in the `<head>` tag of all your webpages: `<script src="behavior.js"></script>`.

Before anything, start by copy-pasting this into your `behavior.js` file:

```
document.addEventListener("DOMContentLoaded", function(event) {  
  
});
```

For this part of the project, all of your JavaScript code must be written between those two lines. Any code outside of these two lines would execute before the HTML page finished loading, so the result could be very random!

To test that your file is properly taken into account by your webpage,

write `alert('Hello!');` between the two lines in `behavior.js`. Refresh the webpage; a dialog box should greet you! Think about removing this line before carrying on, so that your users don't get greeted at each refresh!

Setting up your live JavaScript console in the browser

Every modern browser has a JavaScript console, which allows you to execute pieces of JavaScript code live as your webpage is up. Search online where yours is in your browser (for some browsers, it may require to change your settings first).

Once you've found it, type `alert('Hello!');` in it. The dialogue box should show up. Make sure the console remains open for the rest of the project, as the JavaScript console will also tell you if there are code errors in your .js file.

Inserting a large image in the document

Well, you know how to do that! Find a large image online (at least 800 pixels wide), and use its URL to insert it inside your document, wherever you want. You will notice that even if it is wider than your website, it won't overflow, and will resize nicely, because we set a CSS rule for you, to constrain its size.

Then, we want this image to be small at first; and for this too, we already set a CSS rule for you.

Simply add this attribute to your `` tag: `class="small"` in your HTML code. Refresh, and you see it's much smaller.

The whole point of this part of the project, is that you will use JavaScript to remove this `"small"` class (and therefore the image will grow) or putting it back (therefore it will get back to being small) when the user clicks.

Isolating the HTML element inside a variable

This challenge is all about "listening" to the event "user clicked on this image", and then changing the image's size when it happens; therefore, the first thing to do is to capture this image HTML element in a JavaScript variable that we'll be able to manipulate.

The smartest way to capture one particular HTML element with JavaScript is to give it a unique ID (it's like giving it a first name!), and catch it by giving JavaScript this ID. It's easy to do, you can simply add the attribute `id="smart_thumbnail"` to your `` tag, and there it is, now identifiable!

Then, after refreshing your page, you can run this in your browser's JavaScript console, and you'll see that it indeed returns an element, which is your `` element:

```
document.getElementById("smart_thumbnail");
```

You can copy-paste this line into your `behavior.js` file; but you also have to capture the result in a variable called `thumbnailElement`, so that we can manipulate it more easily later in our JavaScript code. All together, it will look like this:

```
var thumbnailElement = document.getElementById("smart_thumbnail");
```

Catching the click event

Add these lines next in your JavaScript file:

tinegawilfred2@gmail.com

```
thumbnailElement.addEventListener("click", function() {  
  
    // write here  
  
});
```

We did mention before that you could get code to execute on user events; well, code that you write instead of `// write here` will execute whenever the user clicks on your image.

As code that happens when the user clicks on the image, write `alert("I saw you click!");`. Refresh the webpage, then scroll down in your content, and see the dialog box pop up each time you click!

That's great because it means you're catching the click event properly! But a dialog box is not what we want to do, so you should remove your alert statement before carrying on. But note that the rest of your code in this part of the project will be executed when the click is captured, and therefore you will all have to write it there.

Make the image big!

To make the image bigger or smaller, you have to change its class. In your JavaScript console, run this code:

```
var thumbnailElement = document.getElementById("smart_thumbnail");  
thumbnailElement.className;
```

This displays what the class currently is; it should be `"small"`, unless you changed it since.

To make it big, you should remove the class, by running this:

```
thumbnailElement.className = "";
```

To make it small again, you can put it back:

```
thumbnailElement.className = "small";
```

See how it changes from small to big? You should put the line that makes it big in your JavaScript file so that it executes when the user clicks.

One more thing...

After clicking on the image, now that your image is big, your user can... well, do nothing more!

Wouldn't it be cool if the image went back to being small when she clicks again?

This will be an extra challenge for you, if you have some time left after the project!

To make it work, you will need to execute some code *if* the image is big or *else* some other code if not. To that end, you can use what are called an `if` statement and an `else` statement, about which you can [learn more here](#). The `if` statement to check if the image is big should look like this (pay attention to the double-equal sign):

```
if (thumbnailElement.className == "") {  
  
    // write here the code that will execute if the image is big }  
}
```

Once you like what you have, feel free to move on to the next part of the project.

4. DIAQUIS COMMENT

JavaScript is so powerful that it allows to also very simply integrate full-fledged third-party services. [Disqus](#) provides comments for free, very easy to integrate to each of your webpages.

Your project

Find out how to create a Disqus account and add Disqus comments to your site. The integration you're looking for to make it work with a static website such as yours is the one they call "Universal Code". Then, remove the paragraph in your `<aside>` tag, and replace it by your Disqus comment thread.

Once you're getting a nice-looking comment section in your `<aside>` column, and once you're happy about your website overall, feel free to continue to the next project.

5. BE SOCIAL

For this part of your project, you will add social sharing icons to your website, to at least Twitter and Facebook, so that your visitors can share your page if they like it.

First, add a `<div>` tag inside your `<header>` tag on each of your webpages. You should add it first, before the `` tag that contains your navigation.

Add a `class="right"` attribute to your `<div>` tag; we wrote the CSS rule for you so that it will be at the top right of your website.

Next, simply add the social sharing buttons inside that `<div>` tag.

Here are the links to the official tutorials from Twitter and Facebook, but you can add share button to other social websites too if you feel like it:

- <https://developers.facebook.com/docs/plugins/share-button>
- <https://about.twitter.com/resources/buttons#tweet>

Are you done with your project? Now is probably the right time to see if there aren't other ways to make it better, and show off what you can do beyond the constraints of a subject.

You're almost entirely done with level 2; once you're happy with your project, and you feel you're done with it, feel free to move forward: