Front-end

Because websites won’t build themselves

1.0

{LAB ASTEK}

Front-end

Hey! Glad you're here.

As you probably know, web-development is a major branch of software engineering. In fact, a Full-stack developer is the most demanded developer job in 2020.

This workshop is the first part of a series of activities designed to guide you throughout the development of a website, from web page creation to production deployment.

In order to understand what front-end is, you have to familiarize yourself with the basics of web development.

The first thing you should know is that any web application is separated in two parts:

- A back-end, the server, responsible for figuring out **what** data to send to a visitor
- A front-end, the browser, deciding **how** to display the received data

The data is sent to the user in the form of an `.html` file, which contains information about:

- The document structure (HTML), that tells the browser the layout of the elements on a web page
    - Example: "The page should contain a section with a title, two inputs, a select menu and one button"
- The stylesheets (CSS), that define the visual part of each element
    - Example: The text should be centered, the inputs should be 150px wide and the button should have a blue background and white text.
- The scripts (JS), that specify how the elements on the page should behave.
    - Example: once the button is clicked, send a request to the server and update the page contents accordingly.

HTML

CSS

JAVASCRIPT

HTML

Any HTML document is made of two parts: a **head** and a **body**.

HEAD

This part allows you to provide general information about your document. The *title* of the page, its *encoding* and any additional required files you want to include all go in this section.

Fill the *head* tag of your document. Your page must be named "*Epinotes*". Give it a *favicon* located at `img/favicon.png`, a *stylesheet*, `css/master.css`, and a *script* file, `js/index.js`.

BODY

Now let's talk about the *body* of your page. This part describes the main content the document.

Fill your page's *body* with two tags: a <nav> that will hold the navbar and a <div> that will hold the main content.

THE NAVBAR

The <nav> element should contain five links (*a*nchor tags) to our site's pages. For the sake of this example, and since we don't have a real site yet, they should point to "*#*", which will not redirect the user to any page once they click on them.

The first link must contain an image, `logo.png`, that can be found in the `img` folder. If the image fails to load, the browser should display the text "Epinotes" instead.

The remaining four links must have an "*item*" class, and should contain the following text, accordingly: "*Notes*", "*Profile*", "*About*" and "*Log out*".

THE MAIN CONTAINER

The second element in *body* should be a *div* with a "*container*" class.

This *div* must contain:

- A *div* with a "*centered*" class
  - An *input* with an *id* of "*search-input*" and a *placeholder* of "*Search*"
- A *div* with a "*notes*" class
  - Three *div*s, each with a "*note*" class, including:
    - One *<h1>* tag for the title of the note
    - One *<p>* tag for its content

The titles of the notes should be "*Don't panic*", "*So long*" and "*Time is an illusion.*", and you should fill them with the contents of the appropriate `.txt` files located in the `res/` folder at the root of your repository.

CSS

So you've built the inner structure of your webpage, but now it probably looks like some ugly website from the early 90's. That's because it doesn't yet have a *stylesheet*.

Let's fix that!

For this part, you will only need to edit the CSS file located at `css/master.css`. It already contains a few declarations to get you started. Now let's start styling our elements one by one.

EACH NAVBAR ITEM

As for now, the links you put in the nav section of your document look small and compressed. They don't have enough contrast and make it hard to navigate on our site. Here's how we will address that:

- Increase the *size* of their *font* to `14pt`
- Give them a nice white *color*
- Spread them out a bit! Add a *padding* of `23px` on all sides
- Now they look better, but they would look way cooler without that nasty underline. Find a way to remove it!
- If you're really into it, you can even make it *transition* to an 80% *opacity* when *hovered* with the cursor

LAST NAVBAR ITEM

The last item being the "Log out" option, we need to make it stand out from the rest of the *item*s. The good news is, there is a CSS pseudo-class that targets only the *last item* of a selector. Once you find it, give your last *.item* the following styles:

- Make it a little dimmer than its siblings by setting its *color* to `#eee`
- Most importantly, find a way to pin it to the right of the navbar, away from the other items

MAIN CONTAINER

Noticed how our page just sticks to the sides of the browser? Let's space it up a bit:

- Add our .container a `20px` *padding*
- *Align* the *text* of every *.centered* element to… Well, the `center`

SEARCH BAR

Look at that search bar! It looks tiny and unappealing. That's not what we want, is it?

- Set its *font size* to `14pt`
- Now add a *padding* of `10px` on all sides and a **vertical** *margin* of `10px`
- Sets its *width* to `400px` ***min***imum.
- Give it a *solid gray* border of `1px`
- Round its corners in a `5px` *radius*
- Again, if you're really into it, you can add a cool *focus transition*. To do that:
    - Remove the *outline* (this will prevent the default blue glowing effect when *focus*ed)
    - Make it transition to a neat *box-shadow* when focused. A good one might be "`0 0 5px #aaf`"
    - Set the *color* of its *border* to `#aaf` when *focus*ed

NOTES

For now, all our notes look just like some output from the linux manual. It would be great to have them side by side, and also be centered to the middle. There are several methods for achieving that in CSS, but today we will go with the **flexbox** one. Take a moment to research what it is and how to use it. Once you're done, give their parent (.notes):

- The appropriate *display*
- A property to *justify* all its *content* to the `center`

EACH NOTE

By now your notes should be aligned from left to right, with each note taking as much space as it wants. That looks a bit chaotic. Fix that by styling each *.note* as follows:

- Make each note `300px` wide
- Give them a **vertical** and a **horizontal** *padding* of `10px` and `20px` accordingly
- Give them a **vertical** and a **horizontal** *margin* of `20px` and `10px` accordingly
- Give them a *box-shadow*. A nice value might be "`0 0 10px rgba(0, 0, 0, 0.15)`"
- Round their borders in a `20px` *radius*
- Now stretch your browser window to be the size of a mobile device.
  See how the items squeeze? That's what we call poor UX.
  Luckily, you can address that by adding a single property to the flex container (*.notes*), making its children *wrap*. Let's do that!

THE HEADER OF EACH NOTE

Alright, we're almost done. Let's just style the header (*h1*) **of each *.note***:

- Make its *font* a little smaller. `18pt` will do.
- Change that peculiar default *margin*: set it to `5px` at the top and `0` everywhere else (this can be done in one line)
- Set the bottom *padding* to `10px`
- Give it a *bottom border* to separate it from the content. The value doesn't really matter… We figured "`1px solid #ccc`" looked nice

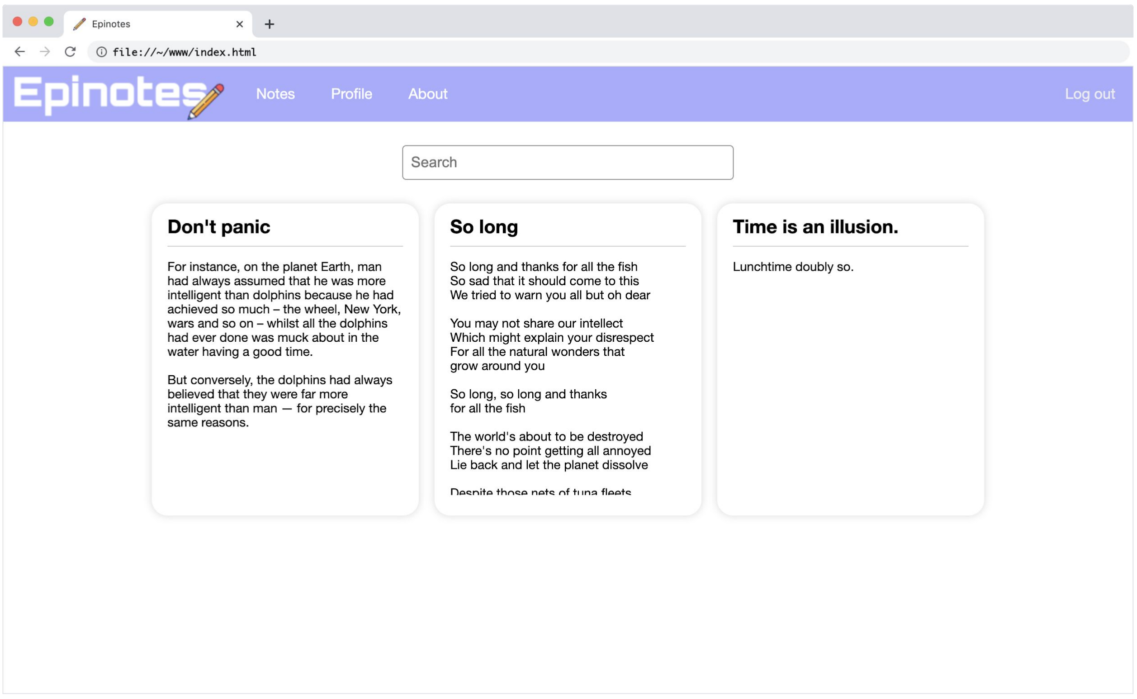
THE CONTENT OF EACH NOTE

Final step! You're 3 lines away from finishing the CSS! Let's conclude by styling the content (*p*) **of each *.note***:

- First, make it `300px` high. You will probably notice that this causes the second note to freak out. We will fix that right away.
- Make the large notes `scrollable` on the *y* axis!
- Finally, you might have noticed that by default html doesn't break lines at the same place as you do. So give the text a single property that will fix that, while also removing unnecessary indentation.

FINAL RESULT

Congratulations, you have just styled your very first HTML webpage! Here's a quick preview of what it should look like, for reference:



Don't panic

For instance, on the planet Earth, man had always assumed that he was more intelligent than dolphins because he had achieved so much – the wheel, New York, wars and so on – whilst all the dolphins had ever done was muck about in the water having a good time.

But conversely, the dolphins had always believed that they were far more intelligent than man — for precisely the same reasons.

So long

So long and thanks for all the fish
So sad that it should come to this
We tried to warn you all but oh dear

You may not share our intellect
Which might explain your disrespect
For all the natural wonders that
grow around you

So long, so long and thanks
for all the fish

The world's about to be destroyed
There's no point getting all annoyed
Lie back and let the planet dissolve

Despite those nets of tuna fleets

Time is an illusion.

Lunchtime doubly so.

JAVASCRIPT

Alright, now that you have created and styled a beautiful search bar, it's about time to actually make it search for things.

For this part, you will be writing code in the file located at `www/js/index.js`. Don't forget to include it in the head section of your document. This file already has some driver code that calls the onLoad function when the document's DOMContentLoaded event fires.

Your job is to fill the `onLoad` function so that it adds an event listener to the search input you have previously created. The event you will be listening for is called `"input"`. This event is triggered every time the user changes the *value* of the *target*ed element, which in our case is the search bar.

The callback for this event should be a function that takes one parameter (the `event` itself) and, according to the value of its *target*, sets the *visibility* of every `".note"` to either `"visible"` or `"collapse"`, depending on whether or not it matches the search query.

A note is considered to match the query if either its title **or** its content includes any word present in the query.

For example, a search for "*fish*" should only leave the second one, and a search for "*Dolphins love fish*" should leave the first two notes (because the word "*dolphins*" is present in the first note, and "*fish*" in the second).

A word is any series of characters separated by spaces.
Have a look at the `String.split()` method!

The search should be **case-insensitive** and an empty query should **not** hide any notes.

Do not use any third-party frameworks or libraries.
This task can be easily done in vanilla JavaScript!

Want to search in the contents of both children of a *.note* at once?
`Node.textContent` is your friend ;)