Introduction to web technologies

HTML + CSS + Javascript

Phillip Ssempeebwa(@philldevcoder)

Introduction

When you decide to develop an application using any programming language, one of the first problem you face is that programming languages do not include a library to create User Interfaces.

You need to use some framework to access the OS layer. Every programming language has at least one, but you need to choose it first.

Your Code
Framework
Operative System
Hardware

Framework and OS

What you expect from your framework to access the OS is usually three things:

- To display information in the screen
- To get the user input
- To request data from the internet.
- To play audio
- To store data
- To get system info like date, screen resolution, etc.

Every programming language has its own set of libraries to fulfill all these tasks but sometimes it can get annoying to set up all of them.

Your Code		
Display	Input	Network
Audio	Storage	System info

The Web Framework

One of the nice things about developing for the web is that the web provides a very rich and simple framework to create applications that include lots of features, not only interface but also access to peripherals (audio, input, gamepads, etc), and this API is very easy to use.

And because this API is universal, it means that it works in every browser of every platform.

Your Code (Javascript)

Browser

| Display | Input | Network |
| Audio | Storage | System info |

Goals

Introduction to web technologies:

- HTML to create the document structure and content

- CSS to control is visual aspect

- Javascript for interactivity

Tools

What do we need to start:
- a good web-browser (Chrome or Firefox)
- a good text editor like:
  - [VSCode](#) (cross platform)
  - [Notepad++](#) (win)
  - textWrangler (osx)
  - [sublime text](#) (cross platform)
  - [ecode](#) (cross platform)
- [the example HTML code](#) to start

How can I test my code

Just open the index.html from the template in your text editor and in your browser.

When you do any change to the code, check it in the browser by pressing F5 (refresh site)

To open the developer tools press:
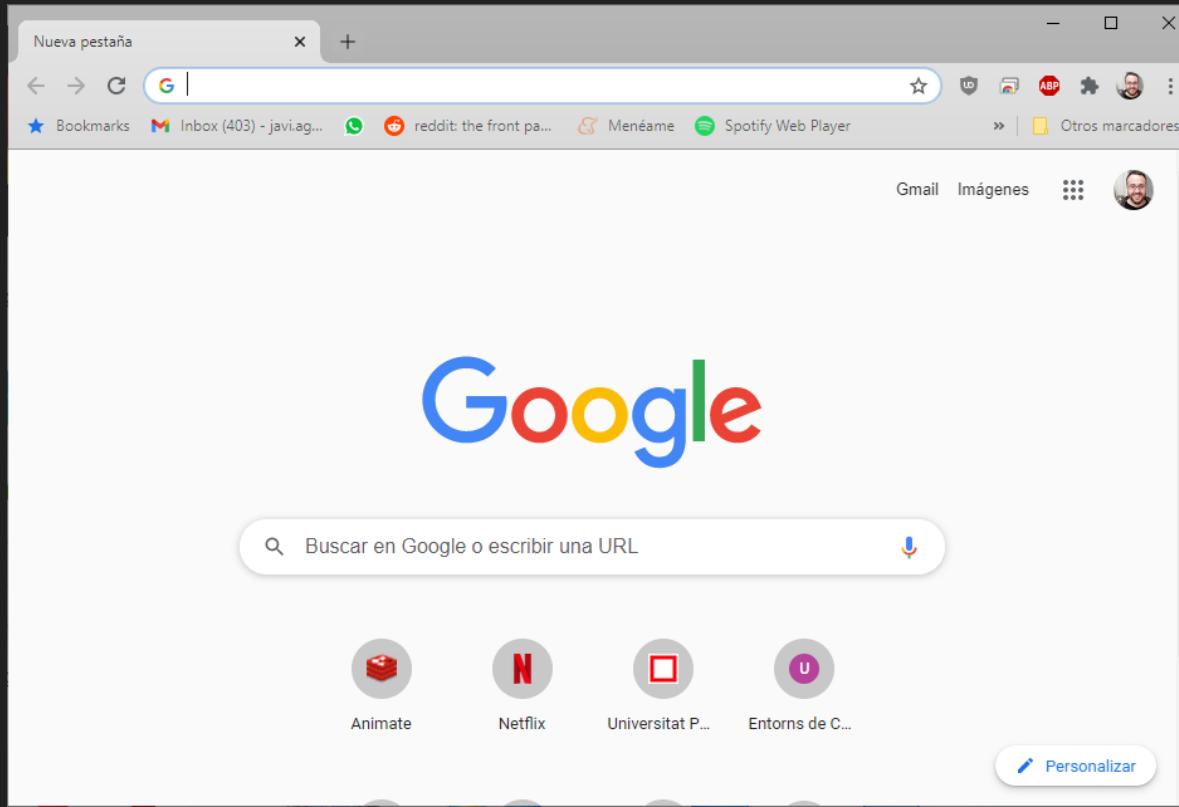
Windows: `Control + Shift + I` or

OSX: `Command + Opt + I`

Other tools are online editors like scratchpad or htmledit

Anatomy of a Browser
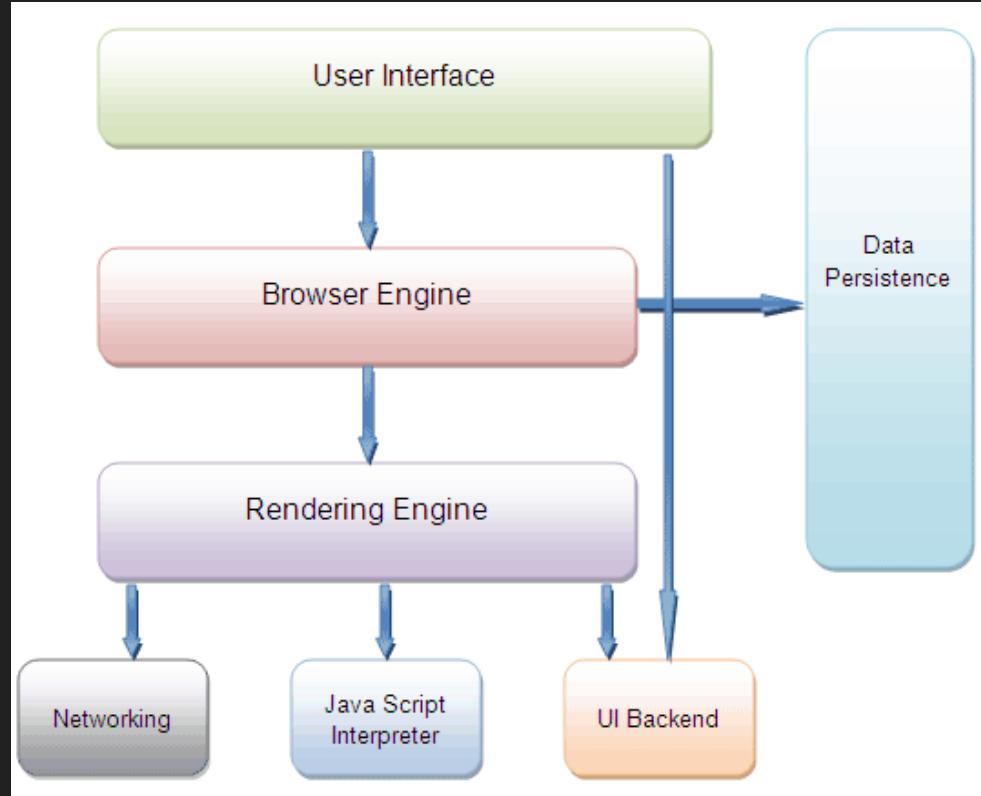
Inside a browser

Browsers have very differentiate parts.
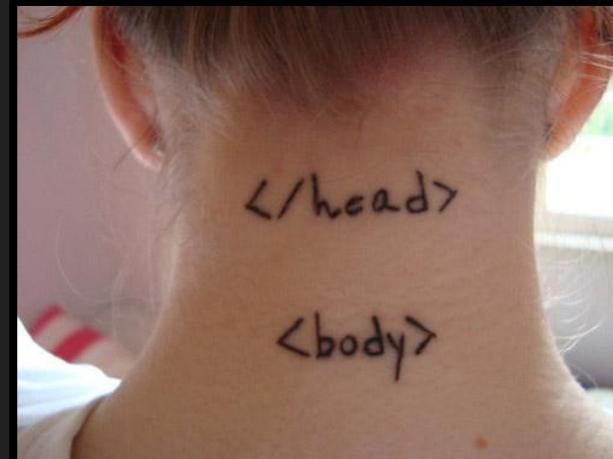
We are interested in two of them:

- the Rendering Engine (in charge of transforming our HTML+CSS in a visual image).
- The Javascript Interpreter (also known as VM), in charge of executing the Javascript code.

Technologies

- HTML

- CSS

- Javascript

Browsers as a renderer
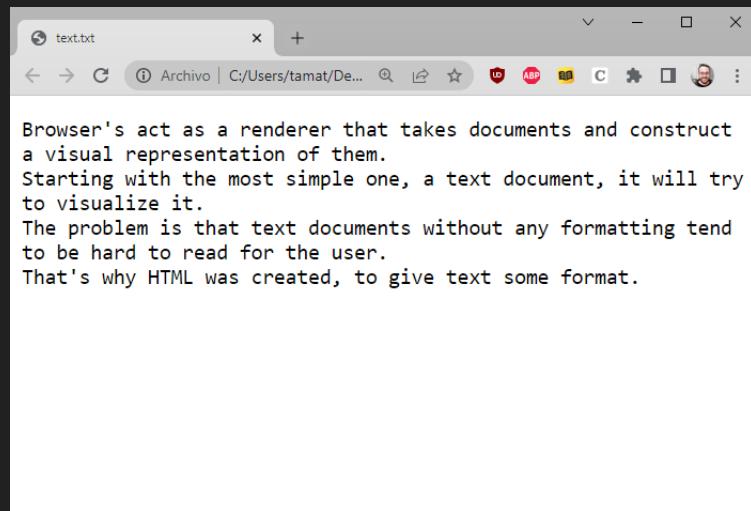
Browser's act as a renderer that takes documents and construct a visual representation of them.

Starting with the most simple one, a text document, it will try to visualize it.

You can try, drop any .txt file into your browser to visualize it.

The problem is that text documents without any formatting tend to be hard to read for the user (and quite boring).

That's why HTML was created, to give text some format.
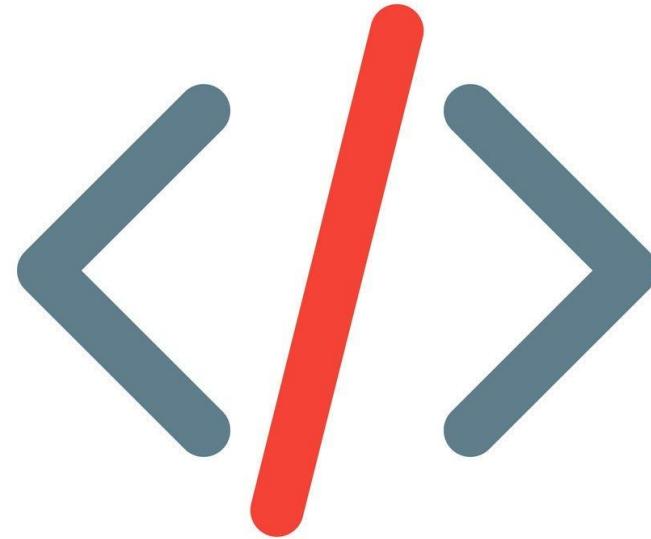
Markup language

There are many markup languages that add special tags into the text that the renderer wont show but use to know how to display the text.

In HTML this tags use the next notation:

```
My name is <b>Javi</b>
```

HTML

HTML means Hyper Text Markup Language.

The HTML allow us to define the structure of a document or a website.

HTML is **NOT** a programming language, it's a markup language, which means its purpose is to give structure to the content of the website, not to define an algorithm.

It is a series of nested tags (it is a subset of XML) that contain all the website information (like texts, images and videos). Here is an example of tags:

```
<title>This is a title</title>
```

The HTML defines the page structure. A website can have several HTMLs to different pages.

```html
<html>
        <head>
        </head>
        <body>
                <div>

        <p>Hi</p>

        </div>
        </body>
</html>
```

HTML: basic rules

Some rules about HTML:

- It uses XML syntax (tags with attributes, can contain other tags).

  `<tag_name attribute="value"> content </tag_name>`
- It stores all the information that must be shown to the user.
- There are different HTML elements for different types of information and behaviour.
- The information is stored in a tree-like structure (nodes that contain nodes inside) called DOM (Document Object Model).
- It gives the document some semantic structure (pe. this is a title, this is a section, this is a form) which is helpful for computers to understand websites content.
- It must not contain information related to how it should be displayed (that information belongs to the CSS), so no color information, font size, position, etc.

HTML: syntax example

```html
<div id="main">
      <!-- this is a comment -->
    This is text without a tag.
    <button class="mini">press me</button>
    <img src="me.png" />
</div>
```

HTML: syntax example

Tag name

attributes
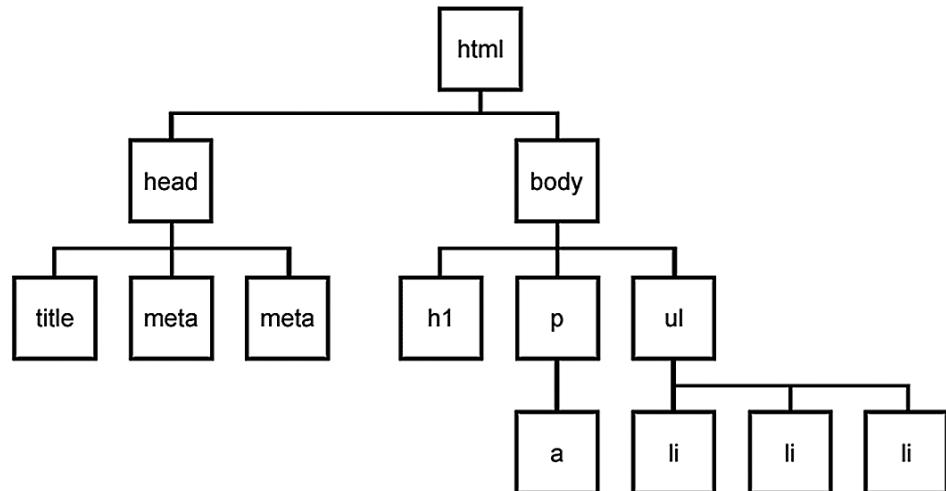
comment

```
<div id="main">
      <!-- this is a comment -->
      This is text without a tag.
      <button class="mini">press me</button>
      <img src="me.png" />
</div>
```
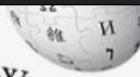
text tag

self-closing tag

DOM is a tree

Every node can only have one parent, and every node can have several children, so the structure looks like a tree.

Article   Talk

Firefox

From Wikipedia, the free encyclopedia

Read Edit View history Search

For other uses, see Firefox (disambiguation)

"Phoenix (web browser)" redirects here. For the Phoenix browser based on tkWWW, see tkWWW.

Mozilla Firefox is a free and open source web browser descended from the Mozilla Application Suite and managed by Mozilla Corporation. As of January 2012, Firefox has approximately 25% of worldwide usage share of web browsers, making it the second or the third most widely used browser, according to different estimates.[11][12][13][14] The browser has had particular success in Germany and Poland, where it is the most popular browser with 52% usage[15] and 45%,[16] respectively.

To display web pages, Firefox uses the Gecko layout engine, which implements most current web standards in addition to several features that are intended to anticipate likely additions to the standards.[16]

Firefox runs on various operating systems including Microsoft Windows, Linux, Mac OS X, FreeBSD, and many other platforms. Its current stable release is version 11.0, released on March 13, 2012.[17]

mozilla
Firefox

Original authors   Mozilla Corporation
Developer(s)   Mozilla Corporation
                    Mozilla Foundation
Initial release   November 9, 2004

Stable release

Preview release

Written in

Operating system

Engine

Type

Contents [hide]

1 History
  1.1 Early versions
  1.2 Version 2.0
  1.3 Version 3.0
  1.4 Version 3.5
  1.5 Version 3.6
    1.5.1 Out-of-process plug-ins
  1.6 Version 4.0
  1.7 Rapid release
    1.7.1 Version 5.0
    1.7.2 Version 6.0
    1.7.3 Version 7.0
    1.7.4 Version 8.0
    1.7.5 Version 9.0
    1.7.6 Version 10.0
    1.7.7 Version 11.0
  1.8 Future releases
    1.8.1 Version 12.0
    1.8.2 Version 13.0
2 Version
  2.1 Released versions
  2.2 Development versions table

HTML: main tags

Although there are lots of tags in the HTML specification, 99% of the webs use a subset of HTML tags with less that 10 tags, the most important are:

- **<div>**: a container, usually represents a rectangular area with information inside.
- **<img/>**: an image
- **<a>**: a clickable link to go to another URL
- **<p>**: a text paragraph
- **<h1>**: a title (h2,h3,h4 are titles of less importance)
- **<input>**: a widget to let the user introduce information
- **<style> and <link>**: to insert CSS rules
- **<script>**: to execute Javascript
- **<span>**: a null tag (doesn't do anything), good for tagging info

HTML: other interesting tags

There are some tags that could be useful sometimes:
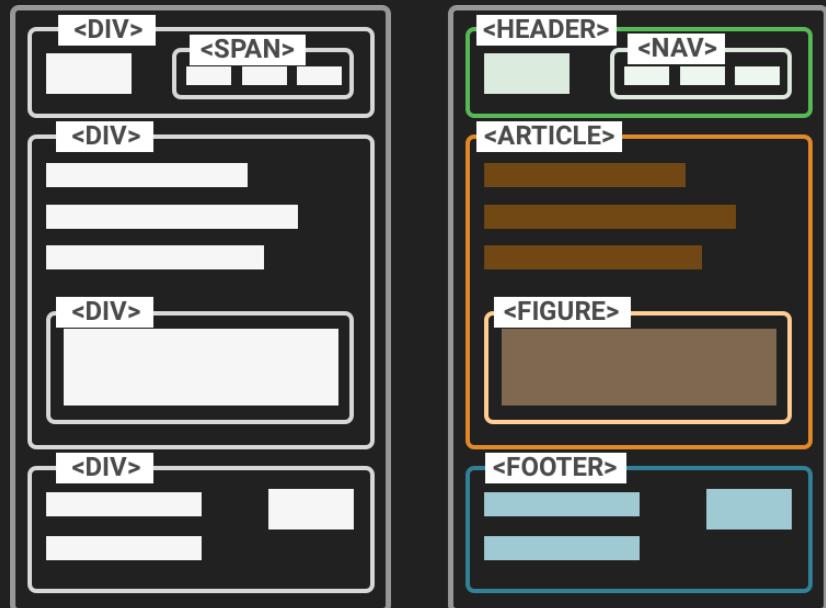
- **\<button\>**: to create a button
- **\<audio\>**: for playing audio
- **\<video\>**: to play video
- **\<canvas\>**: to draw graphics from javascript
- **\<iframe\>**: to put another website inside ours

HTML: wrapping the info

We use HTML tags to wrap different information on our site.

The more structure has the information, the easier will be to access it and present it.

We can change the way the information is represented on the screen depending on the tags where it is contained, so we shouldn't be worried about using too many tags.

HTML: tagging correctly

Try to avoid doing this:

```
<div>
Title

Here is some content
Here is more content
</div>
```

DONT DO THIS

Do this instead

```
<div>
    <h1>Title</h1>
    <p>Here is content.</p>
    <p>Here is more content</p>
</div>
```

HTML good use

It is good to have all the information properly wrapped in tags that give it some semantics.

We also can extend the code semantics by adding extra attributes to the tags:

- id: tells a unique identifier for this tag
- class: tells a generic identifier for this tag
- hidden: to tell the browser to do not show this element

```
<div id="profile-picture" class="mini-image">...</div>
```

HTML references
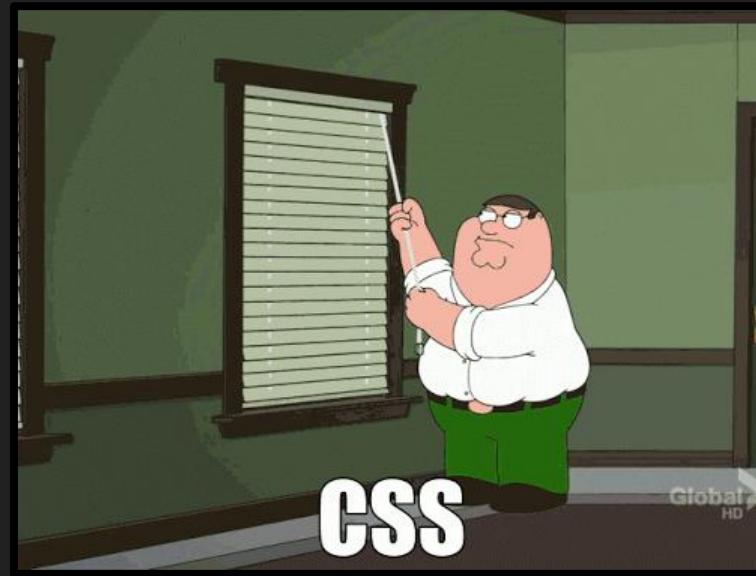
HTML Reference: a description of all HTML tags.

The 25 Most used tags: a list with information of the more common tags.

HTML5 Good practices: some tips for starters

Technologies

- HTML
- CSS
- Javascript
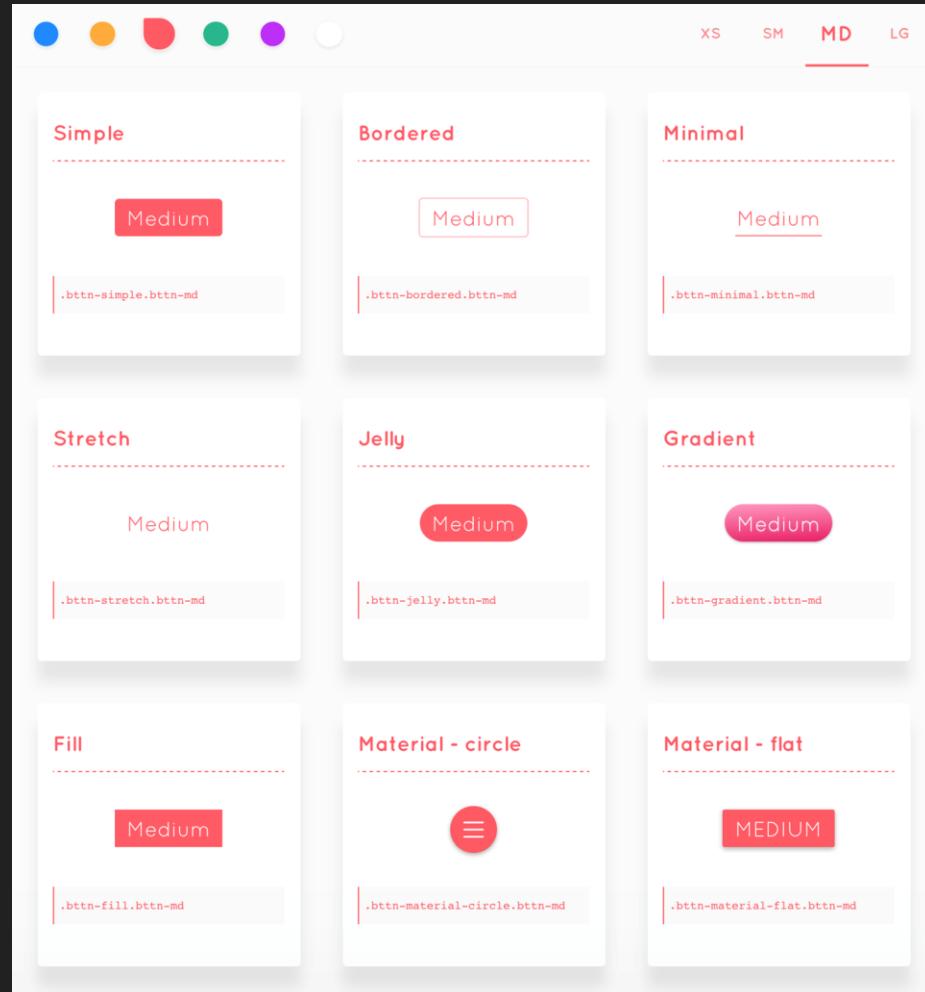
CSS

CSS allows us to specify how to present (render) the document info stored in the HTML.

Thanks to CSS we can control all the aspects of the visualization and some other features:
- **Colors**: content, background, borders
- **Margins**: interior margin, exterior margin
- **Position**: where to put it
- **Sizes**: width, height
- **Behaviour**: changes on mouse over

CSS example

```css
* {
        color: blue; /*a comment */
        margin: 10px;
        font: 14px Tahoma;
}
```

This will change all the tags in my web ('*' means all) to look blue with font Tahoma with 14px, and leaving a margin of 10px around.

CSS fields

Here is a list of the most common CSS fields and an example:

- `color`: #FF0000;   red;   rgba(255,00,100,1.0); //different ways to specify colors
- `background-color`: red;
- `background-image`: url('file.png');
- `font`: 18px 'Tahoma';
- `border`: 2px solid black;
- `border-top`: 2px solid red;
- `border-radius`: 2px; //to remove corners and make them more round
- `margin`: 10px; //distance from the border to the outer elements
- `padding`: 2px; //distance from the border to the inner elements
- `width`: 100%;   300px;   1.3em; //many different ways to specify distances
- `height`: 200px;
- `text-align`: center;
- `box-shadow`: 3px 3px 5px black;
- `cursor`: pointer;
- `display`: inline-block;
- `overflow`: hidden;

CSS how to add it

There are four ways to add CSS rules to your website:

- Inserting the code inside a style tag
  ```
  <style>
      p { color: blue }
  </style>
  ```
- Referencing an external CSS file
  ```
  <link href="style.css" rel="stylesheet" />
  ```
- Using the attribute style on a tag
  ```
  <p style="color: blue; margin: 10px">
  ```
- Using Javascript (we will see this one later).

CSS selectors

Let's start by changing the background color of one tag of our website:

```
div {
        background-color: red;
}
```

This CSS rule means that every tag DIV found in our website should have a red background color. Remember that DIVs are used mostly to represent areas of our website.

We could also change the whole website background by affecting the tag body:

```
body {
        background-color: red;
}
```

CSS selectors

What if we want to change one specific tag (not all the tags of the same type).

We can specify more precise selectors besides the name of the tag. For instance, by class or id. To specify a tag with a given class name, we use the dot:

```css
p.intro {
        color: red;
}
```

This will affect only the tags p with class name intro:

```html
<p class="intro">
```

CSS Selectors

There are several selectors we can use to narrow our rules to very specific tags of our website.

The main selectors are:

- **tag name**: just the name of the tag
  - `p { ... }  //affects to all <p> tags`
- **dot (.)**: affects to tags with that class
  - `p.highlight { ... } //affects all <p> tags with class="highlight"`
- **sharp character (#)**: specifies tags with that id
  - `p#intro { ... } //affects to the <p> tag with the id="intro"`
- **two dots (:)**: behaviour states (mouse on top)
  - `p:hover { ... } //affects to <p> tags with the mouse over`
- **brackets ([attr='value'])**: tags with the attribute attr with the value 'value'
  - `input[type="text"] {...} // affects to the input tags of the type text`

CSS Selectors

You can also specify tags by its context, for example: tags that are inside of tags matching a selector. Just separate the selectors by an space:

```
div#main p.intro { ... }
```

This will affect to the p tags of class intro that are inside the tag div of id main

```
<div id="main">
    <p class="intro">....</p>  ← Affects this one
</div>

<p class="intro">....</p>  ← but not this one
```

CSS Selectors

And you can combine selectors to narrow it down more.

```
div#main.intro:hover { ... }
```

will apply the CSS to the any tag div with id main and class intro if the mouse is over.

And you do not need to specify a tag, you can use the class or id selectors without tag, this means it will affect to any node of id main

```
#main { ... }
```

CSS Selectors

If you want to select only elements that are direct child of one element (not that have an ancestor with that rule), use the > character:

```
ul.menu > li { ... }
```

Finally, if you want to use the same CSS actions to several selectors, you can use the comma , character:

```
div, p { … }  ← this will apply to all divs and p tags
```

HTML arrange
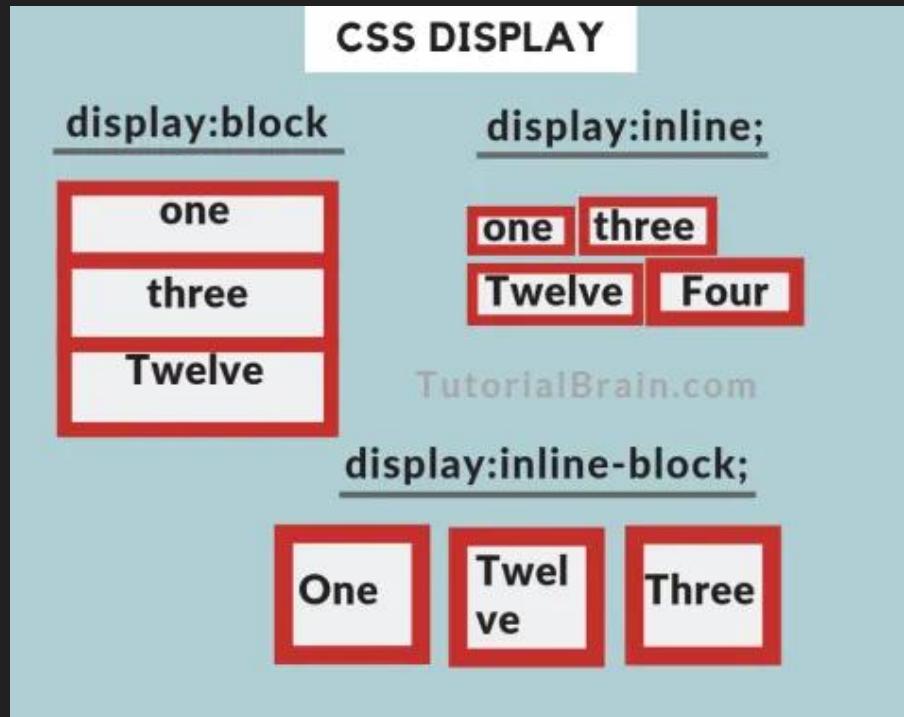
It is important to understand how the browser arranges the elements on the screen.

Check this tutorial where it explains the different ways an element can be arranged on the screen.

You can change the way elements are arranged using the display property:

```
div { display: inline-block; }
```

Also check the property float.
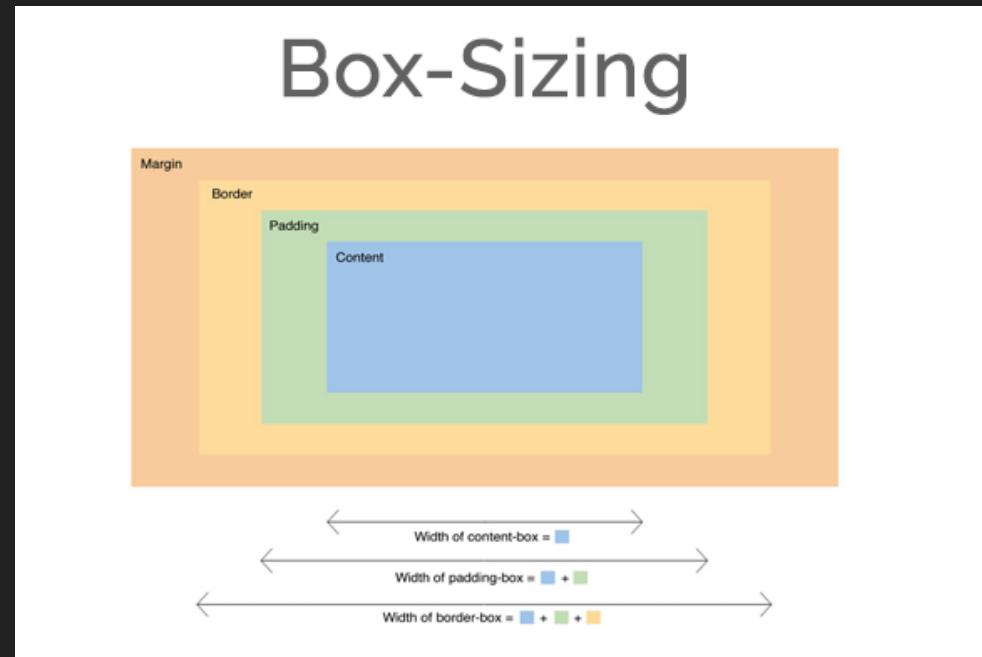
Box Model

It is important to note that by default any width and height specified to an element will not take into account its margin, so a div with width 100px and margin 10px will measure 120px on the screen, not 100px.

This could be a problem breaking your layout.

You can change this behaviour changing the box model of the element so the width uses the outmost border:

```
div { box-sizing: border; }
```
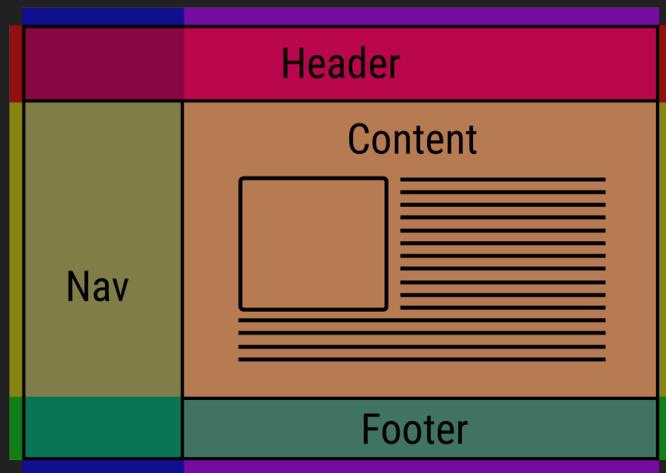
Layout

One of the hardest parts of CSS is construing the layout of your website (the structure inside the window) .

By default HTML tends to put everything in one column, which is not ideal.

There has been many proposals in CSS to address this issue (tables, fixed divs, flex, grid, …).

Flexbox
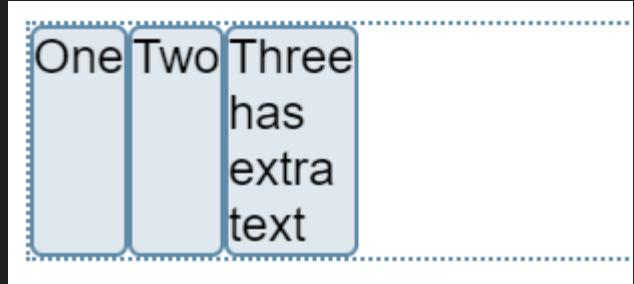
The first big proposal to address the layout was the flexbox model.

This model allows to arrange stuff in one direction (vertically or horizontally) very easily.

You can even choose to arrange from right to left (reverse).

It can also be used to arrange a series of elements in different rows.

Check the tutorial for more info.


```
HTML

<div class="box">
  <div>One</div>
  <div>Two</div>
  <div>Three
    <br>first line
    <br>second line
  </div>
</div>
```
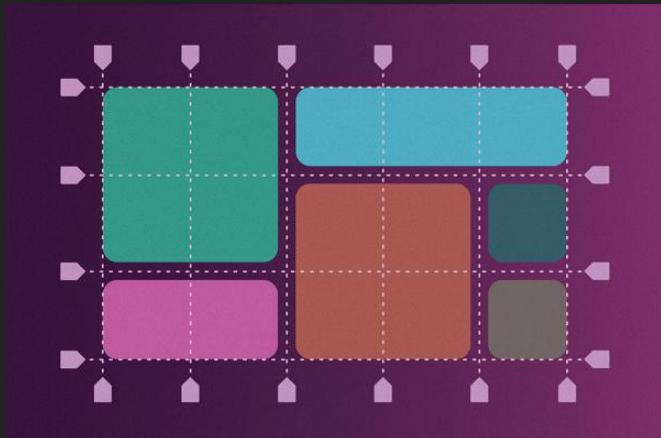
```
CSS

.box {
  display: flex;
}
```

Grid system

Because most sites are structured in a grid, I recommend to use the CSS Grid system.

We just assign how many rows/columns a div should use from the main grid and it will arrange automatically.

Check this tutorial to create the site structure easily


```html
HTML

<div class="grid-container">
  <div class="grid-item1">1</div>
  <div class="grid-item2">2</div>
</div>
```

```css
CSS

.grid-container {
  display: grid;
  grid-template-rows: 100px 100px;
  grid-template-columns: 100px 100px 100px;
  grid-gap: 5px;
}

.grid-item1 {
  background: blue;
  border: black 5px solid;
  grid-column-start: 1;
  grid-column-end: 5;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

Fullscreen divs

Sometimes we want to have a div that covers the whole screen (to make a webapp), instead of a scrolling website (more like regular documents).

In that case remember to use percentages to define the size of elements, but keep in mind that percentages are relative to the element's parent size, so you must set the size to the <body> and <html> element to use 100%.

```css
CSS

html, body {
        width: 100%;
        height: 100%;
}

div {

        margin: 0;
        padding: 0;
}

#main {

        width: 100%;
        height: 100%;
}
```

Trick to center

Centering divs can be hard sometimes, use this trick:

```
.horizontal-and-vertical-centering {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

CSS further reading

There are many more rules for selectors.

Check some of the links to understand them better.

[One line layouts tutorials](#)

[Understanding the Box Model](#): a good explanation of how to position the information on your document.

[All CSS Selectors](#): the CSS selectors specification page.

[CSS Transition](#): how to make animations just using CSS

CSS Templates

There are many existing CSS templates to make your life easier:

- [Bootstrap](#)
- [Bulma](#)
- [TailwindCSS](#)

Here is [a big list](#) with the most famous ones.

Technologies

- HTML

- CSS

- Javascript
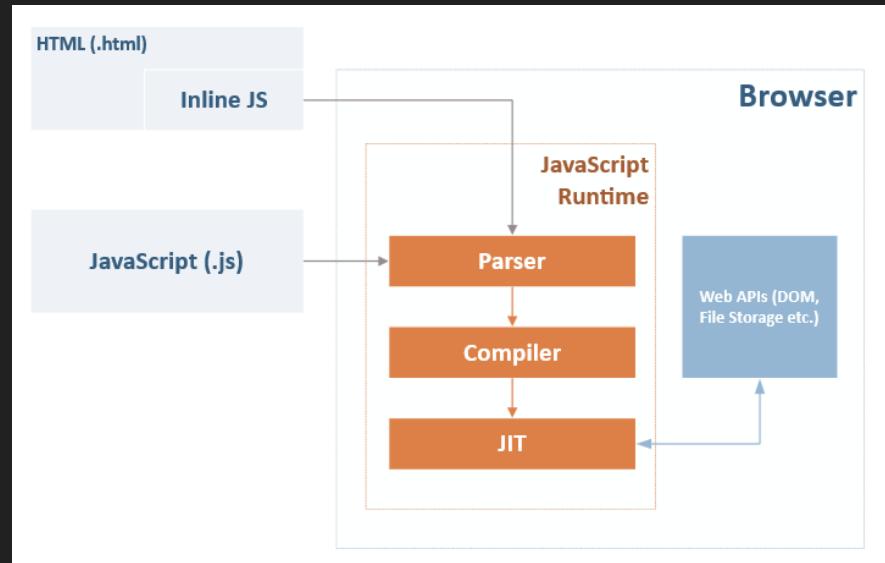
Interactivity

Once the web was already being used they realize people wanted to interact with the websites in a more meaningful way.

So they added an Interpreter to execute a script language that could modify the content of the web dynamically.

Brendan Eich was tasked to develop it in one week and it has become one of the most important languages.

Javascript

A regular programming language, easy to start, hard to master.

Allows to give some interactivity to the elements on the web.

Syntax similar to C or Java but with no types.

You can change the content of the HTML or the CSS applied to an element.

You can even send or retrieve information from the internet to update the content of the web without reloading the page.

```javascript
var my_number = 10;

function say( str )
{
        console.log( str );
}

say("hello");
```

Javascript: insert code

There is three ways to execute javascript code in a website:

- **Embed** the code in the HTML using the <script> tag.

  ```
  <script> /* some code */ </script>
  ```

- **Import** a Javascript file using the <script> tag:

  ```
  <script src="file.js" />
  ```

- **Inject** the code on an event inside a tag:

  ```
  <button onclick="javascript: /*code*/">press me</button>
  ```

Javascript: Syntax

Very similar to C++ or Java but much simpler.

```javascript
var my_number = 10; //this is a comment
var my_string = "hello";
var my_array = [10,20,"name",true];
var my_object = { name: "javi", city: "Barcelona" };


function say( str )
{
        for(var i = 0; i < 10; ++i)
                console.log(" say: " + str );
}
```

Javascript example

```html
<html>
    <body>
        <h1>This is a title</h1>
        <script>
                var title = document.querySelector("h1");
                title.innerHTML = "This is another title";
        </script>
    </body>
</html>
```

Javascript API

Javascript comes with a rich API to do many things like:

- Access the DOM (HTML nodes)
- Do HTTP Requests
- Play videos and sounds
- Detect user actions (mouse move, key pressed)
- Launch Threads
- Access the GPU, get the Webcam image, ...

And the API keeps growing with every new update of the standard.

Check the WEB API reference to know more

Javascript: retrieving element

You can get elements from the DOM (HTML tree) using different approaches.

- **Crawling the HTML tree** (starting from the body, and traversing its children)

- **Using a selector** (like in CSS)

- **Attaching events listeners** (calling functions when some actions are performed)

Javascript: crawling the DOM

From javascript you have different variables that you can access to get information about the website:

- **document**: the DOM information (HTML)

- **window**: the browser window

The document variable allows to crawl the tree:

```
document.body.children[0] // returns the first node inside body tag
```

Javascript: using selectors

You can retrieve elements using selectors:

```javascript
var nodes = document.querySelectorAll("p.intro");
```

will return an array with all `<p class="intro">` nodes in the web.

Or if we have already a node and we want to search inside:

```javascript
var node = mynode.querySelectorAll("p.intro")
```

Javascript: modify nodes

From JS you can change the attributes

```
mynode.id = "intro"; //sets an id
mynode.className = "important"; //set class
mynode.classList.add("good"); //to add to the current classes
```

Change the content

```
mynode.innerHTML = "<p>text to show</p>"; //change content
```

Modify the style (CSS)

```
mynode.style.color = "red"; //change any css properties
```

or add the behaviour of a node

```
mynode.addEventListener("click", function(e) {
        //do something
});
```

Javascript: create nodes

Create elements:
```javascript
var element = document.createElement("div");
```

And attach them to the DOM:
```javascript
document.querySelector("#main").appendChild( element );
```

Or remove it from its parent:
```javascript
element.remove();
```

You can clone an element also easily:
```javascript
var cloned = element.cloneNode(true);
```

Javascript: hide and show elements

Sometimes it may be useful to hide one element or show another.

You can change an element CSS directly by accessing its property style.

To avoid being displayed on the web change display to "none"

```
element.style.display = "none"; //hides elements from being rendered
element.style.display = ""; //displays it again
```

Using Inputs

If you want the user to be able to input some text we use the tag <input>:

```
<input type="text"/>
```

There are other inputs, you can [check this list](#).

From Javascript we can attach events like "click" or "keydown".

To read or modify the content of the input:

```
my_input_element.value = ""; //this will clear the text inside the input
```

Example of a website

HTML in **index.html**

```html
<link href="style.css" rel="stylesheet"/>
<h1>Welcome</h1>
<p>
    <button>Click me</button>
</p>
<script src="code.js"/>
```

CSS in style.css

```css
h1 { color: #333; }
button {
    border: 2px solid #AAA;
    background-color: #555;
}
```

Javascript in **code.js**

```javascript
//fetch the button from the DOM
var button = document.querySelector("button");

//attach and event when the user clicks it
button.addEventListener("click", myfunction);

//create the function that will be called when the
button is pressed
function myfunction()
{

        //this shows a popup window
        alert("button clicked!");
}
```

Execution flow

It is important to have a clear understanding of the execution flow of your code.

Scripts are executed when the html is being parsed.

Be careful accessing the DOM as the DOM won't contain all until all the HTML is parsed.

It is good practice to start your code with an init function called at the end of your HTML.

```html
<script>
        var main =
document.querySelector("#main");
        //main here is null, as the element does
        //exist yet
</script>
<div id="main"></div>
<script>
        var main =
document.querySelector("#main");
        //main now is the right element
</script>
```

jQuery

jQuery is a library that makes working with the DOM much easier, using an unified syntax and taking advantage of selectors:
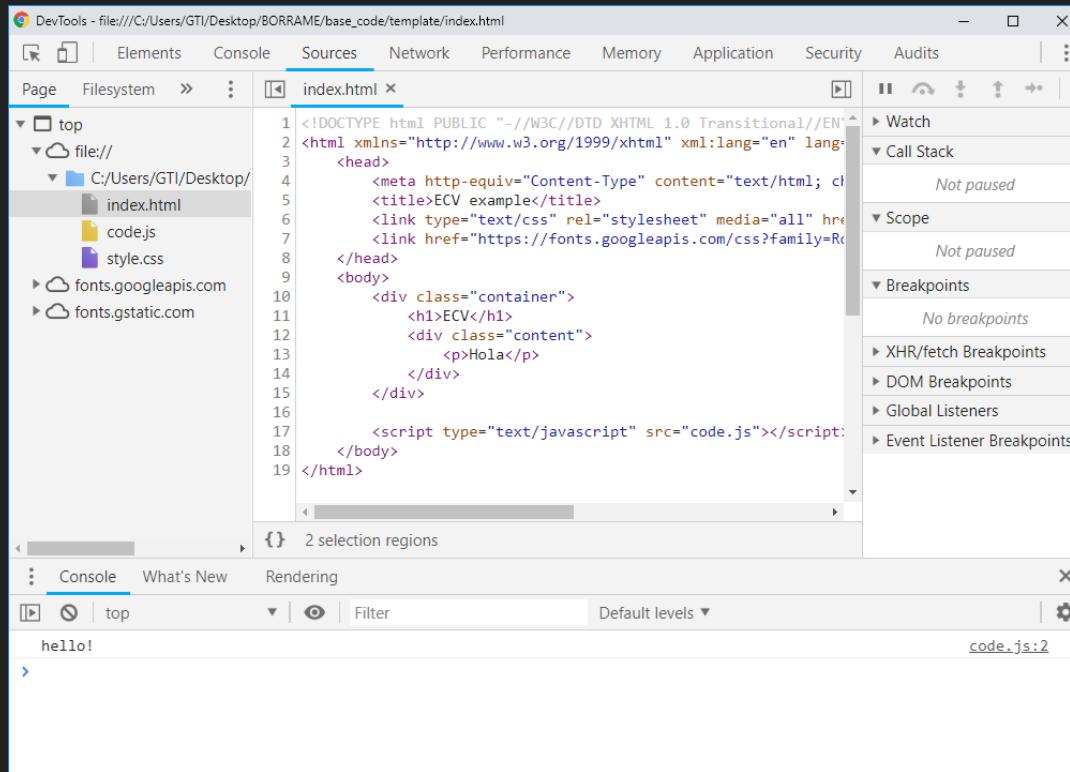
```
$("p").remove(); //remove all tags p
$("#main").hide(); //hides the element of id main
$("#main").append("<h1>titulo</h1>") //adds content to an element

$("#wrap").css({ color: "red" }); //change the css

$("button#send").click( function() { /* code */ });
```

To include this library just add this to your HTML:
```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
```

Using the Dev Tools

Press Control + Shift + I (or F12) to open DevTools
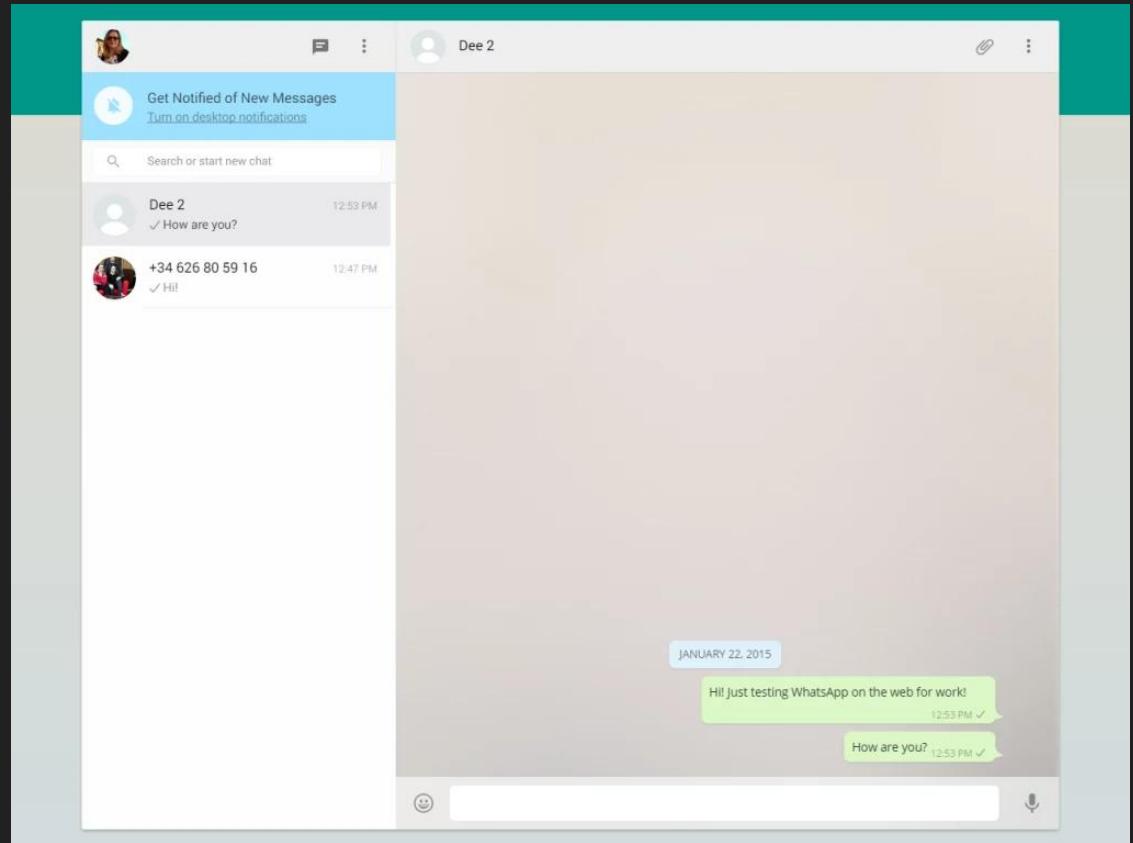
Exercise

Create the layout for a messaging application.

Structured like:

- Main container
  - Messages area
    - message
  - Typing area area
    - input

Further info

API References: DevDocs.io

Selectors: MDN Tutorial

To learn Javascript.

http://codeacademy.com

To learn jQuery:

http://docs.jquery.com/Tutorials