Javascript & CSS

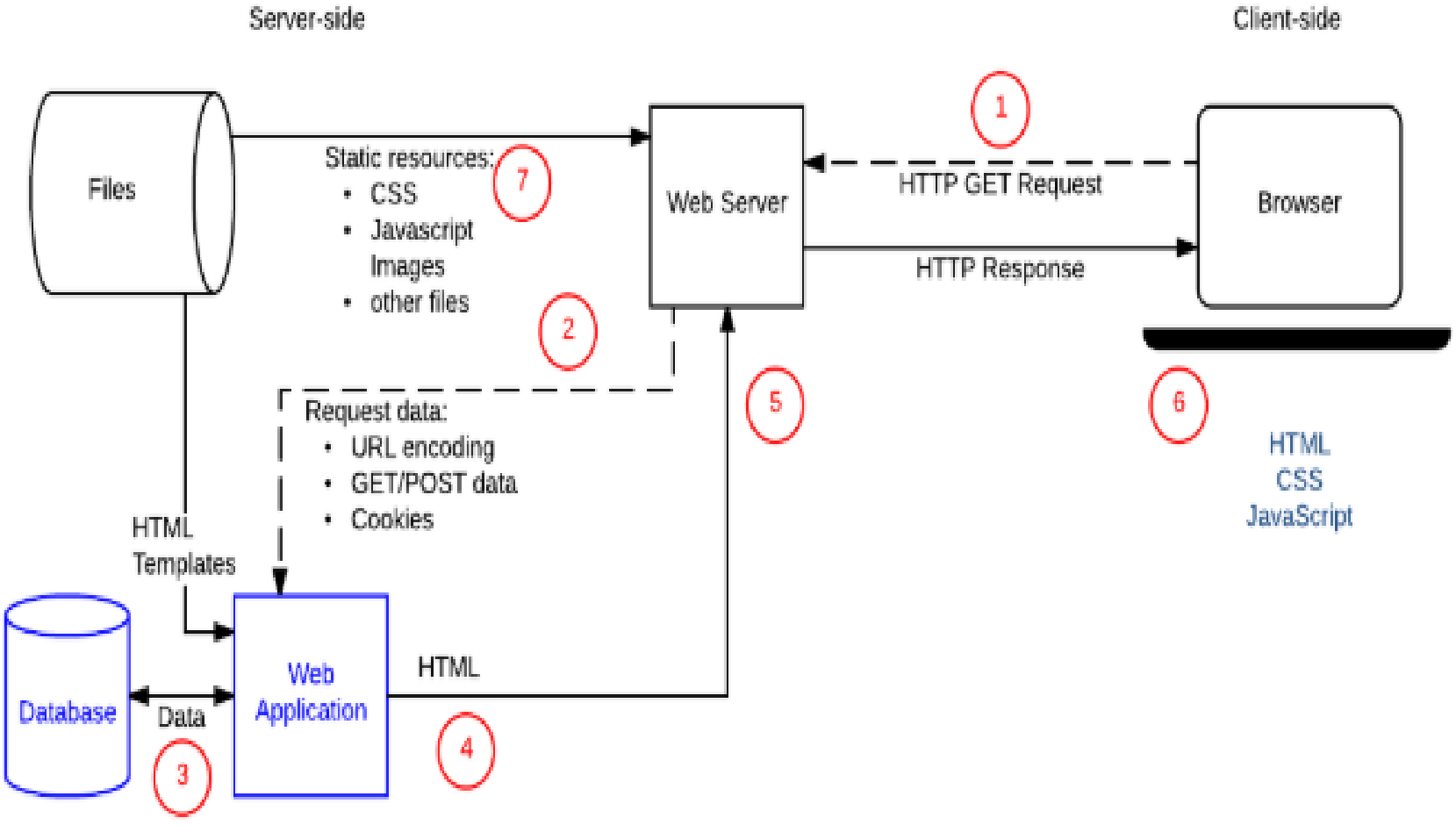
Introduction

- *JavaScript* was initially created to develop interactive/reactive pages.
- Modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.
- JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.
- In-browser JavaScript can do everything related to validation of input, interact with server, handling cookies and time-sensitive events.

Limitations of client-side scripting

- Since script code is embedded in the page, viewable to the world
- For security reasons, scripts are limited in what they can do. e.g., can't access the client's hard drive
- Since designed to run on any machine platform, scripts do not contain platform specific commands.
- Script languages are not full-featured. e.g., JavaScript objects are crude, not good for large project development

Simple architecture for a *dynamic website*



Example

<script> and </script> tells where the JavaScript starts and ends.

```
<html>
<body>
    <h1>My First Web Page</h1>
    <p id="demo">This is a paragraph.</p>
    <script type="text/javascript">

    </script>
</body>
</html>
```

Common HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Functions and Events

```
<!DOCTYPE html>
<html><head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "changed.";
}
</script>
</head>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try</button>

</body>
</html>
```

JavaScript Popup Boxes

- Alert Box - An alert box is often used if you want to make sure information comes through to the user.

```
alert("I am an alert box!");
```

- Confirm Box - A confirm box is often used if you want the user to verify or accept something.

```
if (confirm("Press a button!")) {
        txt = "You pressed OK!";
} else {
        txt = "You pressed Cancel!";
}
```

- Prompt Box - A prompt box is often used if you want the user to input a value before entering a page.

```
var person = prompt("Please enter your name", "Harry Potter");

if (person == null || person == "") {
  txt = "User cancelled the prompt.";
} else {
  txt = "Hello " + person + "! How are you today?";
}
```

JavaScript Timing Events

The two key methods to use with JavaScript are:

- **setTimeout(function, milliseconds)** - Executes a function, after waiting a specified number of milliseconds.
- **setInterval(function, milliseconds) -** Same as setTimeout(), but repeats the execution of the function continuously.

setTimeout() Method

```
<!DOCTYPE html>
<html>
<head>
    <script>
        function myFunction() {
            alert('Hello');
        }
    </script>
</head>
<body>
    <p>Wait 3 seconds, and the page will alert "Hello".</p>
    <button onclick="setTimeout(myFunction, 3000);">Try it</button>
</body>
</html>
```

setInterval() Method

```
<!DOCTYPE html>
<html>
<head>
    <script>
        function myFunction() {
            alert('Hello');
        }
    </script>
</head>
<body>
    <p>The page will alert Hello every 3 seconds.</p>
    <button onclick="setInterval(myFunction, 3000);">Try it</button>
</body>
</html>
```

Client-side Validation

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title></title>
    <script type="text/javascript">
        function validate() {
            if (document.getElementById("txtName").value == "") {
                return false;
            }
            else {
                return true;
            }
        }
    </script>
</head>
<body>
    <form onsubmit="return validate()">
        Name: <input type="text" id="txtName" /><br/>
        <input type="submit" value="SUBMIT" />
    </form>
</body>
</html>
```

jQuery

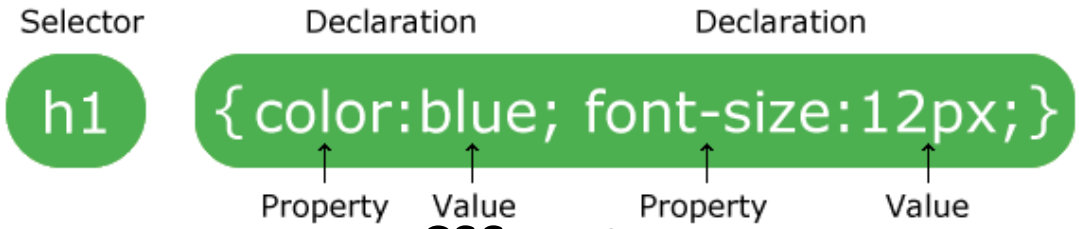
- jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=“
crossorigin="anonymous"></script>
```

```
$(".checkButton").click(() => {
      if ($("#txtName").val() == "") {
          alert('please fill all values');
          return false;
      }
      else {
          return true;
      }
  });
```

Cascading Style Sheets (CSS)

- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once



CSS syntax

Selector	Example	Example description
#id	#firstname	Selects the element with id="firstname"
.class	.intro	Selects all elements with class="intro"
element.class	p.intro	Selects only <p> elements with class="intro"
*	*	Selects all elements
element	p	Selects all <p> elements
element,element,..	div, p	Selects all elements and all <p> elements

How To Add CSS

There are three ways of inserting a style sheet (see examples for each):

- External CSS
- Internal CSS
- Inline CSS

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

!important rule in css is used to override

CSS Selectors

A CSS selector selects the HTML element(s) you want to style.

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

CSS Selectors - Combinator selectors

- descendant selector (space)

```
div p {
  background-color: yellow;
}
```

- child selector (>)

```
div > p {
  background-color: yellow;
}
```

- adjacent sibling selector (+)

```
div + p {
  background-color: yellow;
}
```

- general sibling selector (~)

```
div ~ p {
  background-color: yellow;
}
```

CSS Selectors - Pseudo-classes selectors

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}
```

CSS Selectors - Pseudo-elements selectors

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

```
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}

p::first-line {
  color: #0000ff;
  font-variant: small-caps;
}

::selection {
  color: red;
  background: yellow;
}
```

CSS Selectors - Attribute Selectors

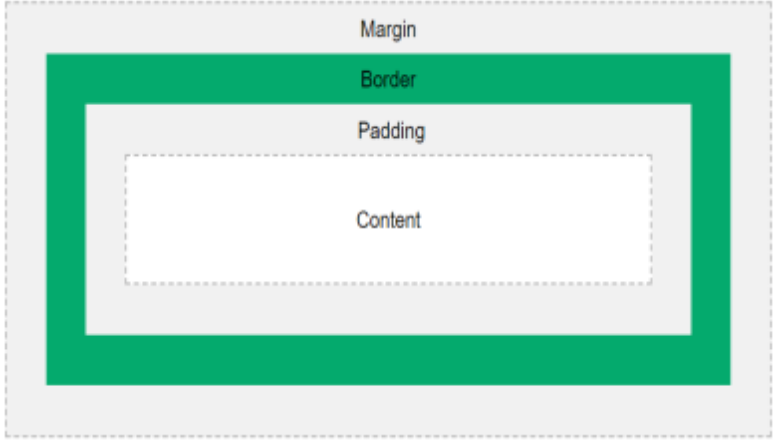
Style HTML elements that have specific attributes or attribute values

```
a[target="_blank"] {
  background-color: yellow;
}
```

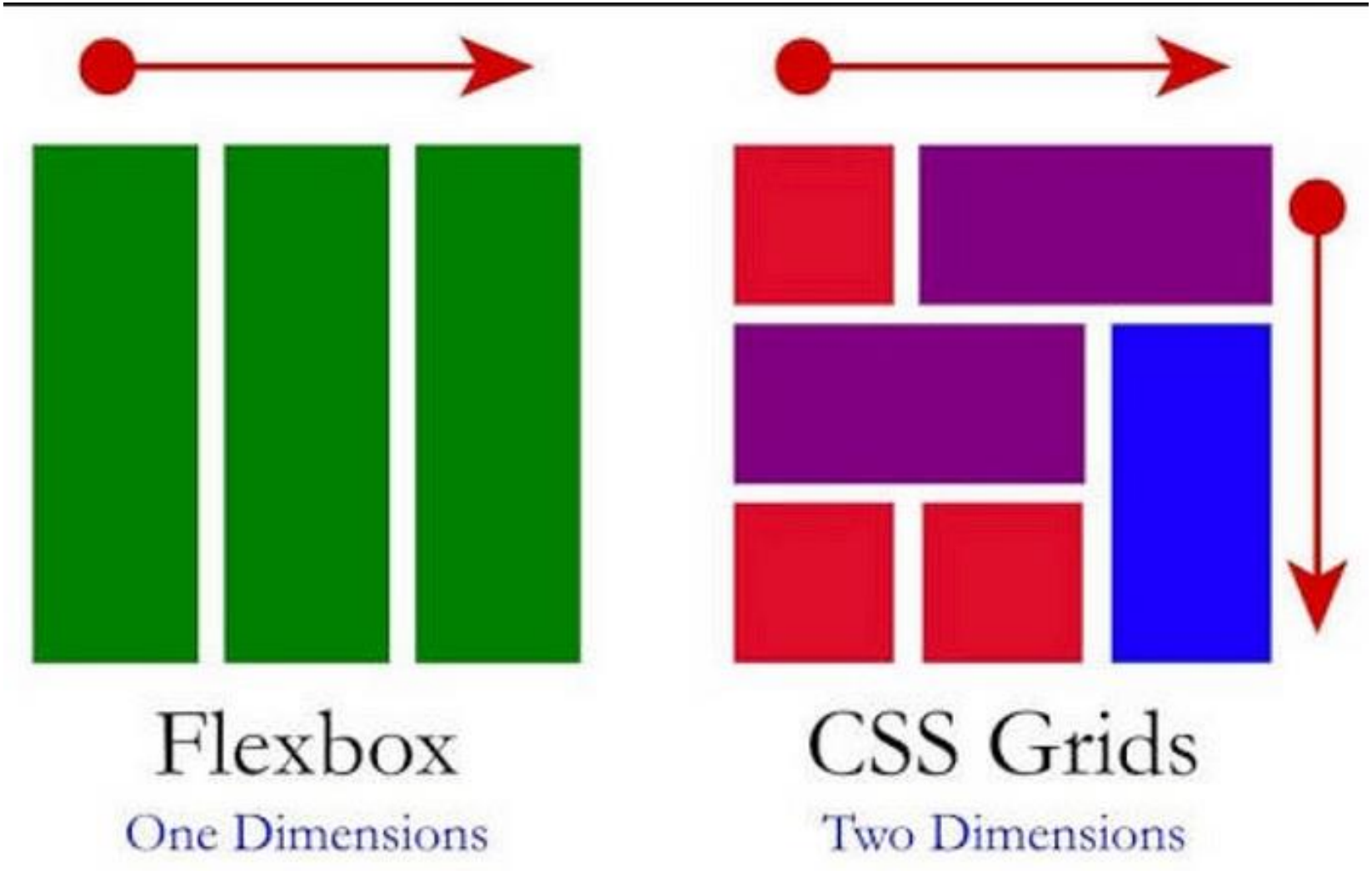
CSS basic box model

- When laying out a document, the browser's rendering engine represents each element as a rectangular box according to the standard CSS basic box model. CSS determines the size, position, and properties (color, background, border size, etc.) of these boxes.

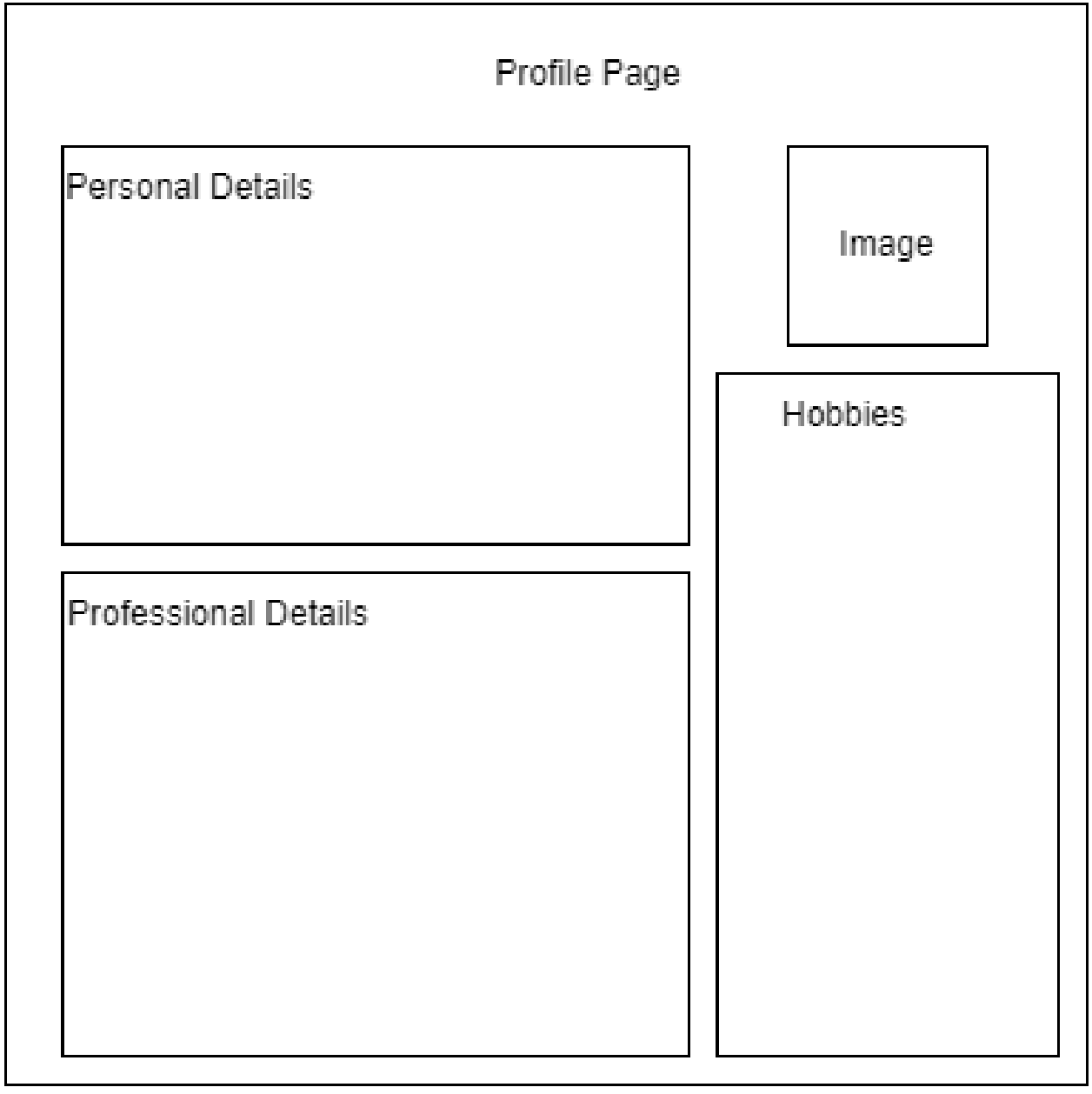
- Every box is composed of four parts (or areas):
  - **Content** - The content of the box, where text and images appear
  - **Padding** - Clears an area around the content. The padding is transparent
  - **Border** - A border that goes around the padding and content
  - **Margin** - Clears an area outside the border. The margin is transparent



CSS Flex vs Grid



ASSIGNMENT –Only HTML+CSS+Js



References

- https://www.w3schools.com/