Web Development: Frontend Fundamentals

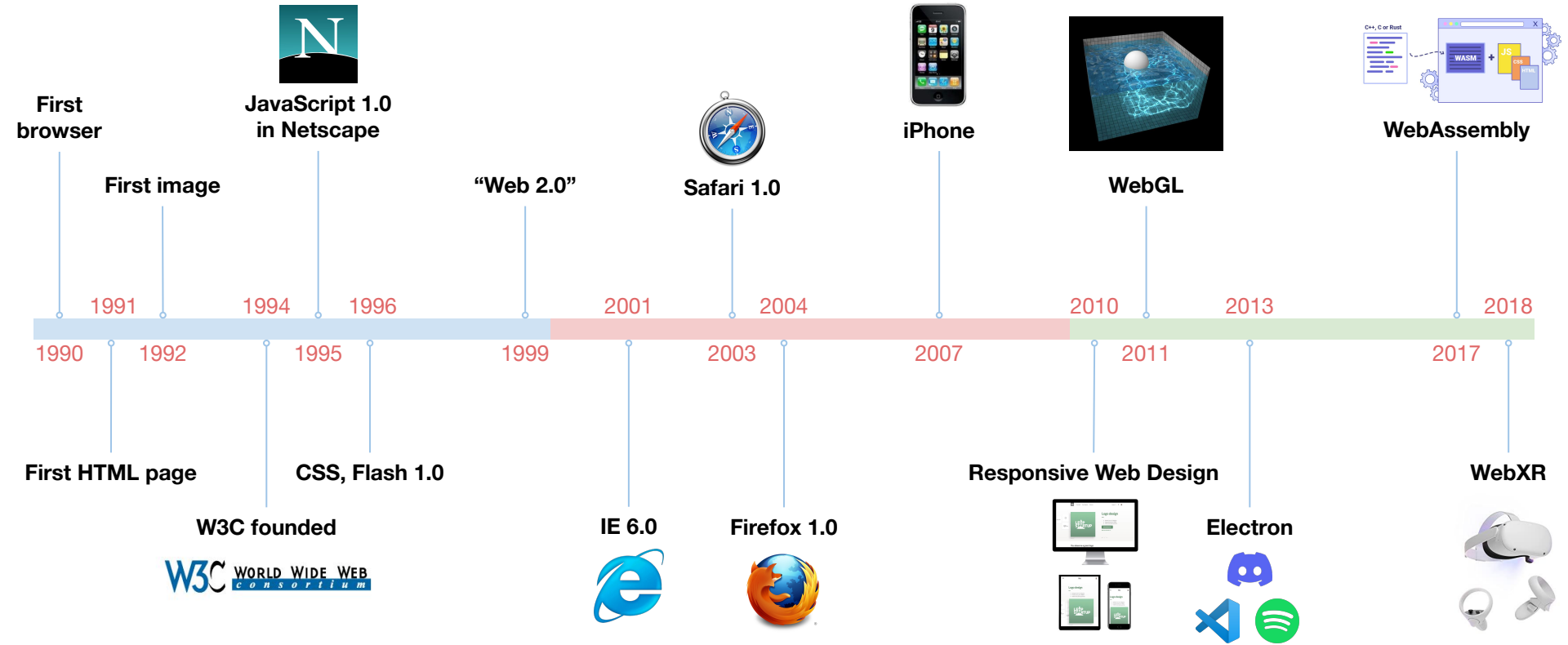
EE 461L Software Engineering & Design
Fall 2022

Evan King
e.king@utexas.edu

Agenda

- **Context:** Brief history of the web
  - From static pages to dynamic applications
- **Deep dive:** Fundamental technologies
  - HTML
  - CSS
  - JavaScript


HeY n WeLcOmE 2
timehop
Abe's Page
since 1995
cOOl LiNkS
Timehop Web Page
aBoUt AbE
Geocities (MaKe Ur OwN WEBSIGHT!!)
SiGN mY GuEsT bOOk
Gallery
eNjOy Ur StAy
UpDaTeD SePt. 12, 1999 !!
u are visitor number
000000034
email the WeBmAsTeR
cute_abe_95@hotmial.com


First browser
First image
JavaScript 1.0 in Netscape
“Web 2.0”
Safari 1.0
iPhone
WebGL
WebAssembly
1991
1994
1996
2001
2004
2010
2013
2018
1990
1992
1995
1999
2003
2007
2011
2017
First HTML page
CSS, Flash 1.0
W3C founded
IE 6.0
Firefox 1.0
Responsive Web Design
Electron
WebXR

From Websites to Web Applications

- In the mid to late 90s, browsers began integrating technologies like JavaScript, which developers used to build more dynamic websites

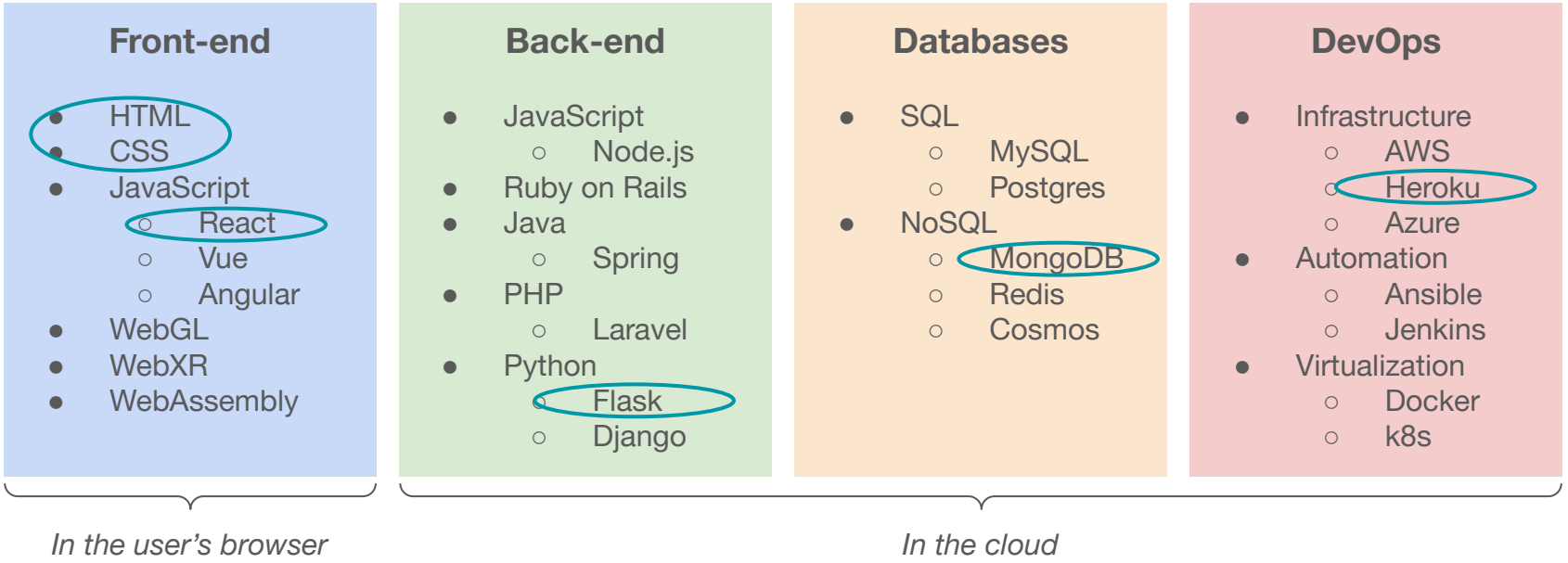
- By 1999, these highly-dynamic websites were termed “web applications”

- Adoption of increasingly powerful web standards by browser vendors has made web applications a popular alternative to native applications

Demo

The “Full Stack”

- **Full-stack web developer:** someone who commands broad knowledge of the full range of web technologies used in modern web application development.



Today’s example: a dynamic website

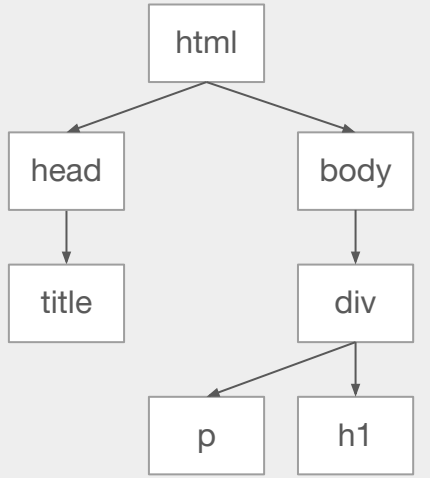
HTML → CSS → JavaScript

HTML (HyperText Markup Language)

The layout of a page. HTML *tags* are used to create *elements* which together define page structure.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Welcome to HTML</title>
    </head>
    <body>
        <div>
            <h1> Welcome! </h1>
            <p> Writing HTML is fun and easy.
</p>
        </div>
    </body>
</html>
```

Document Object Model (DOM)



Tags

`<html>`	Root element of an HTML document
`<head>`	Holds machine-readable metadata about the document
`<title>`	Defines page title (goes in `<head>`)
`<body>`	The visible body of the document
`<div>`	A generic container – the building block of a layout
`<h1>, <h2>, … <h6>`	Text headings (larger numbers = smaller headings)
`<p>`	A paragraph of text
`<a>`	“Anchor” tag, used for links
`<span>`	A generic container for inline text
`<form>`	An interactive area for entering/submitting information
`<input>`	Defines various types of user input
`<button>`	A button

HTML Elements Reference

Attributes

HTML elements have *attributes*:

```
<tag attribute="value"></tag>
```

We configure elements using attributes, like setting this link:

```
<a href="website.com">Link<a>
```

Or setting different input types:

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

We also assign a *class* and/or *id* to elements using attributes:

```
<a class="nav-link"></a>
<button id="submit-btn"></button>
```

Create the layout…

CSS (Cascading Style Sheets)

The design and aesthetics of a page. CSS *selectors* set the *properties* of HTML elements to alter their appearance.

Basic selectors:

```
tag

.class

#id
```

Multiple:

```
.class1, .class2
```

selects all elements of class1 and all of class2

Descendent:

```
div .class1
```

selects all elements of class1 inside a div

???

```
#foo .class1, .class2
```

selects all of class1 inside element id foo and all of class2

[CSS Reference]

```
selector  {
   property: value
}
```

Internal vs. External Stylesheets

Styles are either defined *internally* (inline with HTML) or *externally* (in a separate file) External is usually better: you reuse one design across pages.

Internal:

```
<style>
   p {
       color: hotpink;
   }
</style>
```

Or as an attribute:

```
<p style="color: hotpink;"></p>
```

External:

```
<html lang="en">
  <head>
      <link href="./styles.css" rel="stylesheet">
  </head>
…
</html>
```

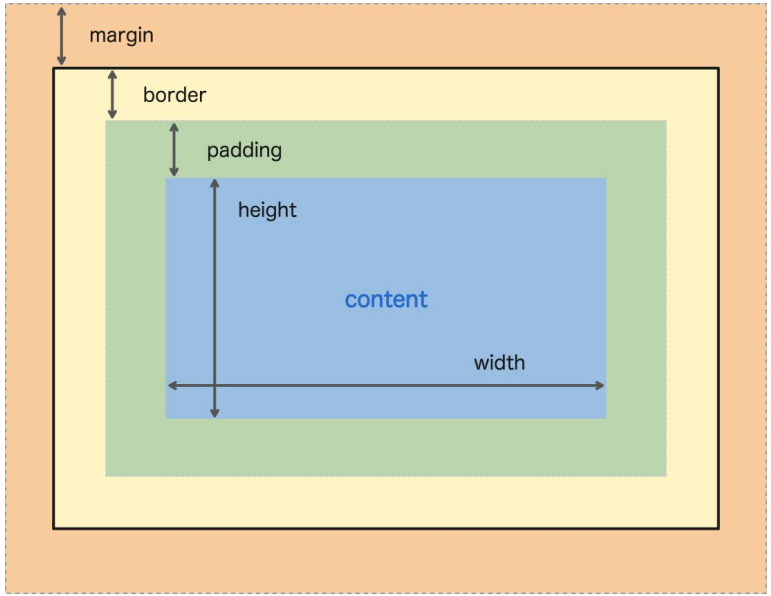
→ styles.css

```
p {
    color: hotpink;
}
```

Box Model Properties

The box model helps us define the positioning and flow of content.

```
div {
    /* top right bottom left */
    margin: 12px 0px 6px 0px;
    border: ...
    padding: ...
    height: ...
    width: ...
}
```



There are a lot of different units of measurement, some absolute (e.g. px) and some relative (e.g. %): CSS Units

Properties

Property	Example Value	Description
`margin`	`12px 0px 6px 0px`	Sets the element's top, left, bottom, and right margin. Margin is *outside* the border
`border`	`1px solid black`	Sets border, as above
`padding`	`16px`	Sets padding, as above. Padding is *inside* the border
`height`	`600px`	"Sets" the element's height. Tends to be finicky
`width`	`60%`	Sets the element's width
`background`	`white`	Shorthand for all background properties (color, image, etc)
`font`	`12px "Comic Sans MS"`	Shorthand for all font properties (size, font face, line height, etc)
`text-align`	`center`	Sets alignment of the text content inside an element (left, right, center)

Responsive Design

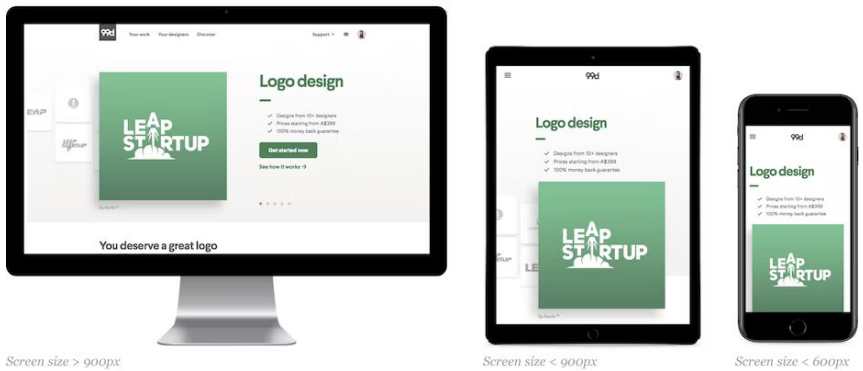
Leverage CSS features to adapt a single page layout to different screen sizes.

Standard CSS:

- Grid + Flexbox layouts

Frameworks:

- Bootstrap
- Pure.css
- etc.

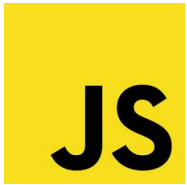


Screen size > 900px *Screen size < 900px* *Screen size < 600px*

Style the page…

JavaScript

The behavior of a page.



- Dynamic types, first-class functions
- Multi-paradigm
  - Functional, imperative, and event-driven programming
- Compiled and run just-in-time by the JavaScript engine
  - In browser or standalone
- ECMAScript is the standard

```
function greet(name) {
   let greeting = 'hello, ' + name
   alert(greeting) // displays a pop-up window
}
```

JavaScript Reference

Integrating with HTML

Scripts can be defined *inline* with HTML or *externally* (in a separate file)

Inline:

```
<script>
   alert('hello')
</script>
```

External:

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

Element Interface

`element` – an interface representing an HTML element

- `element.innerHTML` – returns the HTML inside the element
- `element.getAttribute(attr)` – returns the value of the passed attribute
- `element.append(element)` – appends the passed element after the element's last child
- `element.remove()` – removes the element from the page

Document Interface

`document` – an interface representing the current page in the browser

`document.title` – the title of the page

`document.URL` – the url of the page

`document.getElementById(id)` – returns HTML element with the passed id

`document.createElement(tag, options)` – creates an HTML element

HTML

```
<div id="hello-div">
   Hello!
</div>
```

JS

```
let helloDiv = document.getElementById('hello-div')
alert(helloDiv)
alert(helloDiv.innerHTML)
```

a reference to the element

the element's HTML contents

Event Binding

HTML elements provide an attribute for *binding* events. Events are bound to JavaScript functions which define what happens when those events occur.

Event	Example	Fires when…
onload	`<body onload="..."></body>`	the page loads
oninput	`<input type="text" oninput="..."></input>`	input is entered
onchange	`<select onchange="...">...</select>`	the dropdown value changes
onkeydown	`<input type="text" onkeydown="..."></input>`	a key is pressed down
onclick	`<button onclick="...">click me</button>`	the element is clicked

HTML Event Attribute Reference

Make it dynamic…

Reference Materials

There are a lot of web development resources online. These are (in my opinion) the best of them.

MDN Web Docs - References for HTML, CSS, JS, WASM, Web APIs, etc

Guides on CSS-Tricks - Modern CSS in-depth

Javascript.info - JavaScript in-depth

W3Schools - Straightforward examples for many web technologies