SENG 365 Week 6

Security and Intro to client side

This week

- Primer on Security issues
- Introduction to client-side technologies and concepts

- Assignment 1 queries

Primer on Security for Web Apps

https://owasp.org

https://www.meetup.com/OWASP-New-Zealand-Chapter-Christchurch/

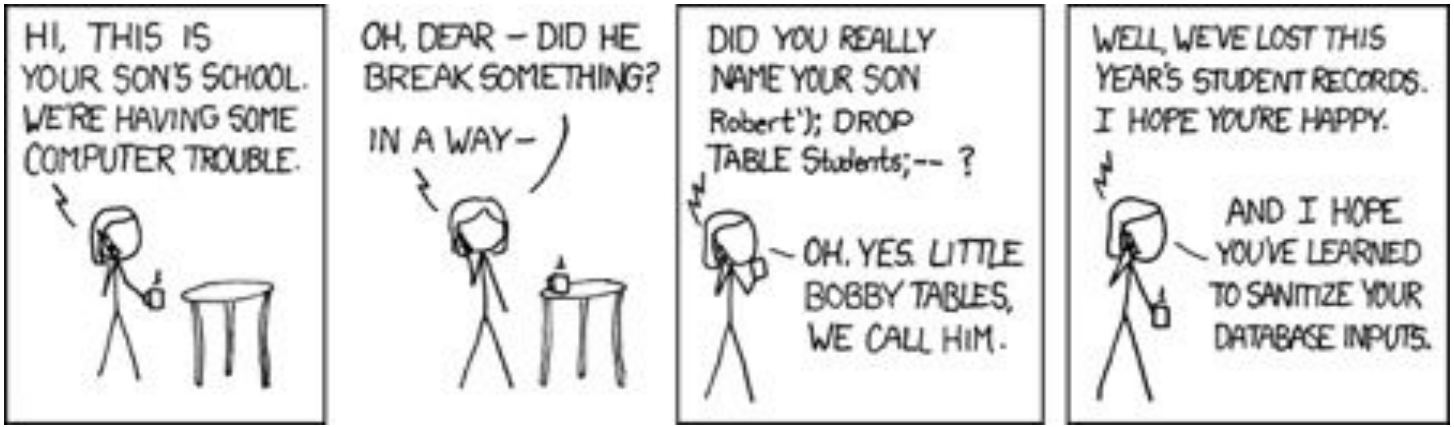
Open Web Application Security Project (updated 2020)

Top 10 security problems
1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

Injection

Injection flaws allow attackers to relay malicious code through an application to another system e.g. SQL injection.

https://www.owasp.org/index.php/Injection_Flaws



Injection

- **Any time** an application uses **an interpreter of any type** there is a danger of introducing an injection vulnerability.
- When a web application passes information from an **HTTP request** through as part of an external request, it must be carefully scrubbed
- **SQL injection** is a particularly widespread and dangerous form of injection...

Command Injection

- Assume that we have a Java class (on the server) that gets input from the user via a HTTP request, and that class goes on to use the Java Runtime object to make an MS-DOS call

```
Runtime rt = Runtime.getRuntime();
// Call exe with userID
rt.exec("cmd.exe /C doStuff.exe " +"-" +myUid);
```

Command Injection

```
Runtime rt = Runtime.getRuntime();
// Call exe with userID
rt.exec("cmd.exe /C doStuff.exe " +"-" +myUid);
```

When `myUid = Joe69`, we'd get the following OS call:

```
> doStuff.exe -Joe69
```

When `myUid = Joe69 & netstat –a`, we'd get:

```
> doStuff.exe -Joe69
> netstat –a // "&" is command appender in MS-DOS
```

Basis of all injections...

- All injection flaws are **input-validation** errors.
  - i.e. you're not checking the input properly
- Input is **not just text fields**
- All **external input** is a source of a **threat**.
  - The input contains the data with the threat
  - Examples: text fields, list boxes, radio buttons, check boxes, cookies, HTTP header data, HTTP post data, hidden fields, parameter names and parameter values

Validate ... and re-validate

- An input field is likely to be **validated on the client side**, e.g., that an IDNumber textfield contains a number rather than a number and malicious code.
  - What happens to that data **between the client and the server**?
- Sometimes it's **hard to validate**.
- Sometimes there are **multiple clients** e.g. you're offering a public web service to clients.
- Should you safely assume that the input data is valid because it was **previously validated**?

Authentication

- Definitions
  - **Authentication**: establish claimed identity
  - **Authorisation**: establish permission to act
  - Authentication precedes authorisation
- **Why** authenticate?
  - Control access to resources
  - Log user activity
  - Non-repudiation
- **How** can we authenticate?
  - Three factors: something you know, have, or are

Securing passwords

- Hash username and password
- Require users to change their passwords regularly
- Use multi-factor authentication
  - Username & password
  - Code sent by phone
- Salt the username and password
  - Add additional elements to the ID information
- Use HTTPS (HTTP + TLS)

HTTP is a “stateless” protocol

- Means credentials have to go with every request
- Should use SSL for everything requiring authentication

Session management flaws

- SESSION ID used to track state since HTTP doesn’t
  - and it is just as good as credentials to an attacker
- SESSION ID is typically exposed on the network, in browser, in logs, …

Beware the side-doors

- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

Typical Impact

- User accounts compromised or user sessions hijacked

Verify your architecture

- Authentication should be simple, centralized, and standardized
- Use the standard session id provided by your container
- Be sure SSL protects both credentials and session id at all times

Verify the implementation

- Forget automated analysis approaches
- Check your SSL certificate
- Examine all the authentication-related functions
- Verify that logoff actually destroys the session
- Use OWASP's WebScarab to test the implementation

Follow the guidance from

- https://www.owasp.org/index.php/Authentication_Cheat_Sheet

DOM-based XSS Injection

- DOM Based XSS allows an attacker to use the **Document Object Model** to introduce **hostile code** into vulnerable client-side JavaScript embedded in many pages.
- **Browser interprets** .js, HTML, the DOM etc
- DOM based XSS is extremely difficult to mitigate against because of its **large attack surface** and **lack of standardization** across browsers.
  - **Untrusted data** should only be treated as **displayable text**. **Never** treat untrusted data **as code** or markup within JavaScript code.
  - Always **JavaScript encode** and **delimit untrusted data** as quoted strings when entering the application

OWASP Tutorials

Learn more here (or take SENG 406)

https://owasp.org/www-project-web-security-testing-guide/

Summary of server-side

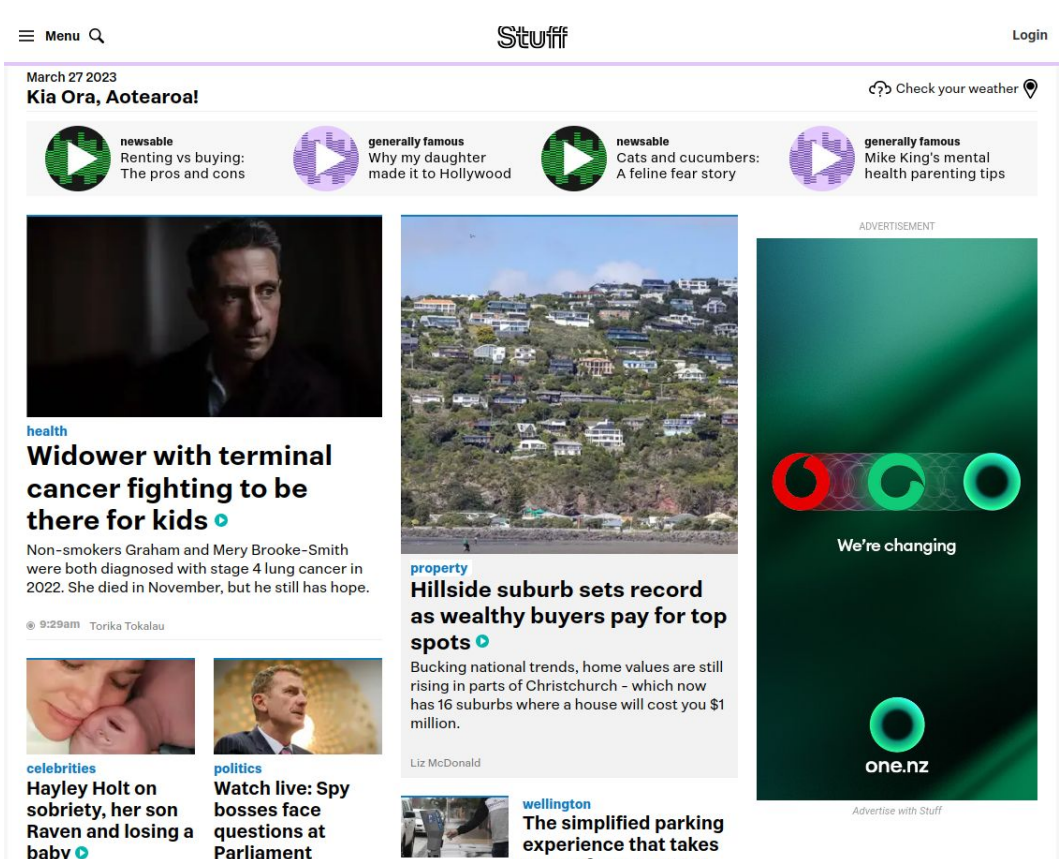
Lectures

- HTTP requests & responses
- APIs, endpoints & API-driven design
- The server itself
- e.g. Node.js & express & node packages
- Data persistence e.g. MySQL

Labs and additional tutorials

- JavaScript, TypeScript
- Node.js + Express
- Data persistence
- APIs
- GraphQL
- OWASP

Introduction to Client-side



What kind of contents ‘appear’ in the browser?

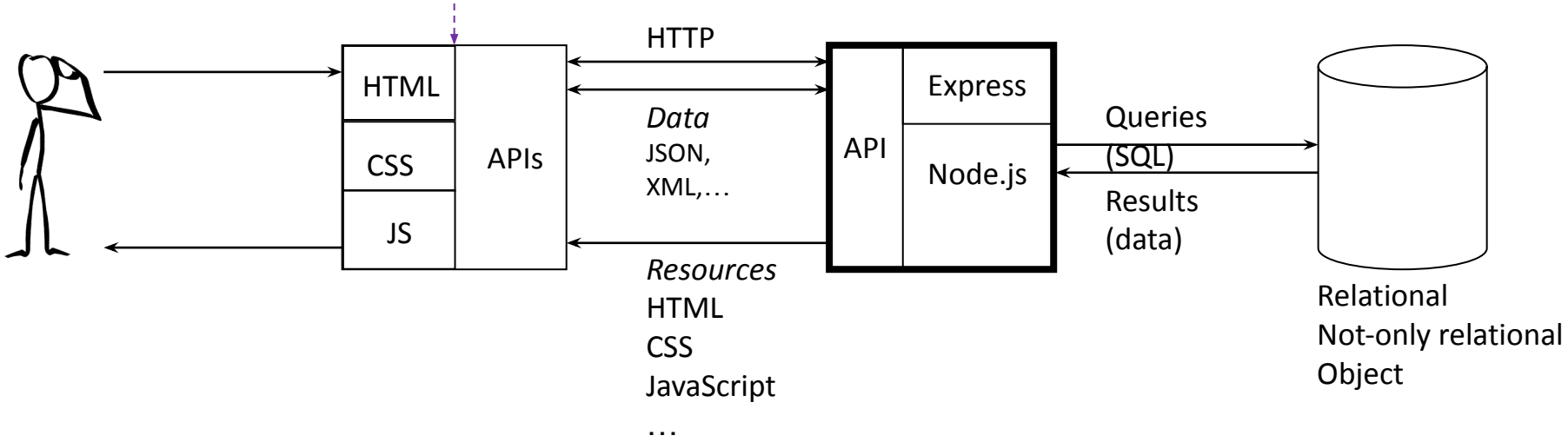
1. What the user sees?
2. How the content is described? (‘view source page’)

How is that content presented?

- Colours, styles, etc.

What is the user experience in the browser?

What does the client-side app need to do to assemble the data from the API endpoints into coherent, interactive 'pages' for the user?



User	Client	Server	Database
Human	Machine	Machine	Machine

Introduction to the main client-side technologies

Client-side: JavaScript

But note...

"Unlike most programming languages, the [core|original] **JavaScript** language has **no concept of input or output**. It is **designed to run** as a scripting language **in a host environment**, and it is up to the **host environment** to **provide mechanisms* for communicating with the outside world**."

https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

* e.g. more APIs

... the full quote:

“Unlike most programming languages, the **JavaScript** language has **no concept of input or output**. It is **designed to run** as a scripting language **in a host environment**, and it is up to the **host environment** to **provide mechanisms for communicating with the outside world**.

The most common host environment is the browser, but JavaScript interpreters can also be found in a huge list of other places, including Adobe Acrobat, Adobe Photoshop, SVG images, Yahoo's Widget engine, server-side environments such as Node.js, NoSQL databases like the open source Apache CouchDB, embedded computers, complete desktop environments like GNOME (one of the most popular GUIs for GNU/Linux operating systems), and others.”

https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

JavaScript + browser !== JavaScript + Node.

With JS + browser:

- You're dealing with a user!
- Input and output via HTML & CSS (& DOM)
- Deployment of your app is different:
  - Which browser, or version.
  - Network-connection quality.
  - Whether cookies are enabled.
  - The computing power of the hosting machine.
- Different APIs in browser
  - AJAX (XHR)
- Dependencies on libraries
  - 'importing' JS libraries is different
    - e.g. CDN vs npm install
  - What about node packages...?
- Project structure is different
  - JS, HTML, CSS, other assets
- Similar terminology, different meaning
  - e.g. "routes" & "routing"

Client-side: HTML

What is HTML?

“**HTML** (HyperText Markup Language) is the most basic building block of the Web. It describes and defines the **content** of a **webpage**. Other technologies besides HTML are generally used to describe a webpage's appearance/presentation (CSS) or functionality (JavaScript).”

(https://developer.mozilla.org/en-US/docs/Web/HTML)

- Be careful about the difference between ***content*** and ***data***.
- Also, be careful about your concept of a **webpage**. What constitutes a ‘web page’ has changed over time.

Yeah, okay, but what is HTML?

- HTML is a **declarative 'language'**, comprising:
- A declaration of a document type (HTML), together with a *hierarchical* structure of (nested) HTML elements, where
  - **elements** are identified by **tags**, and
  - elements typically contain some kind of **content** (to display), and where
  - elements may have **attributes**, in which
  - attributes define **characteristics** of elements,
  - attributes often have **values** (for the characteristics),
  - attributes allow **cross-referencing** to CSS and JavaScript, and
  - attributes may be **custom-defined**.

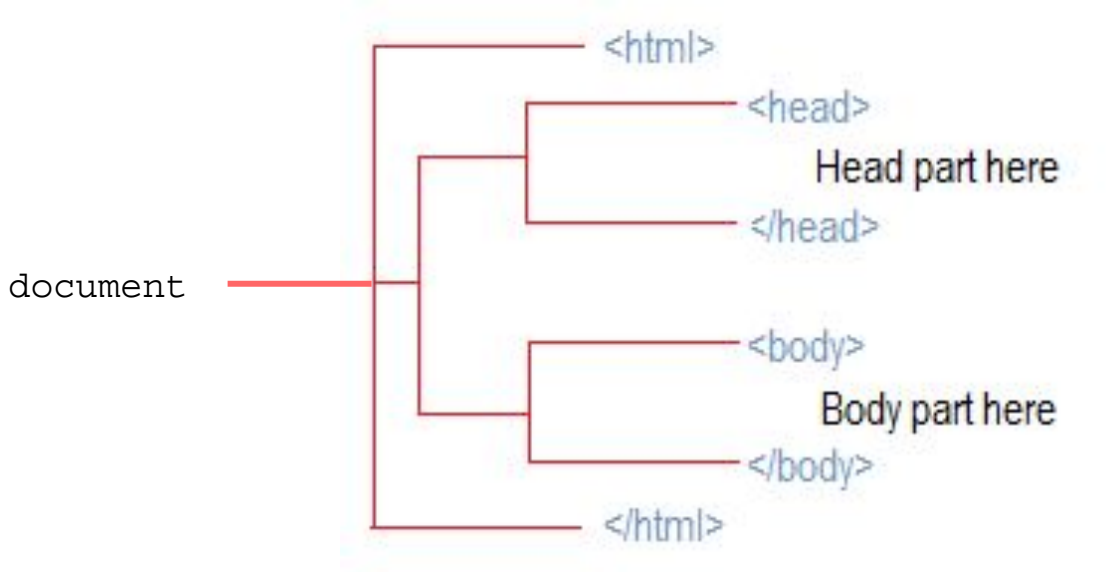
(HTML5 introduced many new elements)

Basic example of HTML page

HTML comprises:

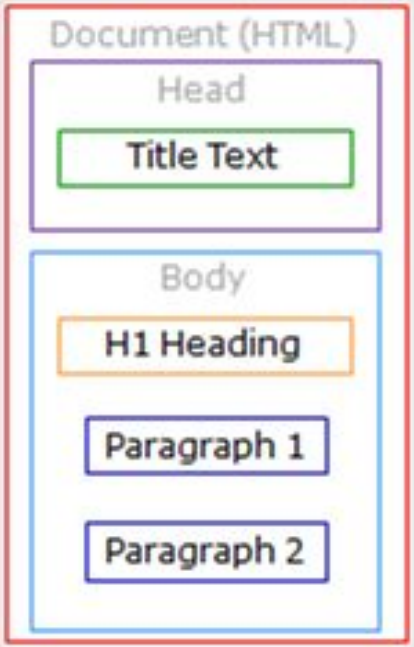
- Document type declaration
- Elements, organised into a hierarchy, e.g.
  - <html>
    - <head>
    - <body>
- Attributes of elements, with values
  - `lang="en"`

```
<!DOCTYPE html>
<html lang="en">
    <head>
            <meta charset="UTF-8">
            <title>A title</title>
    </head>
    <body>

    </body>
</html>
```



http://www.corelangs.com/html/introduction/img/html-page-structure.png

```
<HTML>

 <HEAD>
  <TITLE>Title Text</TITLE>
 </HEAD>

 <BODY>
  <H1>H1 Heading</H1>
  <P>Paragraph 1</P>
  <P>Paragraph 2</P>
 </BODY>

</HTML>
```



HTML elements

For example, if we wanted to write the following on a web page:

Javascript callbacks are turtles all the way down.

```
<p>Javascript callbacks are turtles all the way down.</p>
```

Opening tag

Content

Closing tag

Element

HTML elements & their attributes

For example, if we wanted to write the following on a web page:

Javascript callbacks are turtles all the way down.

```
<p class="comment">Javascript callbacks are turtles...</p>
```

Name

Value

Attribute

Nested HTML elements

For example, if we wanted to write the following on a web page:

Javascript callbacks are **turtles** all the way down.

```
<p>Javascript callbacks are <strong>turtles</strong> ...</p>
```

Nested element

Content

Opening tag

Closing tag

One way to change HTML content: JavaScript

HTML: before

```
<p id="four">Oh, cruel world.</p>
```

JavaScript

```
document.getElementById("four").innerHTML = "Hello world!";
```

HTML: after

```
<p id="four">Hello world!</p>
```

Custom attributes (we'll come back to this)

- You can define your own attributes for elements
- HTML5 offers `data-*` attribute
  - Where * is a string of characters of your choice
  - But potential for name clashes with other JavaScript libraries
    - I define `data-student` in my `student.js` library, and
    - You define `data-student` in your `super-student.js` library
- Frameworks use custom-defined attributes as part of their two-way binding 'magic'
  - Angular has `ng-*` attributes
  - Vue has `v-*` attributes

Pointers elsewhere

Validate your HTML at:
https://validator.w3.org/

Further information on **HTML5** elements:
https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

Client-side: CSS

A (declarative) ‘language’ for specifying how contents are **presented** to a user.

What is CSS?

A set of rules for specifying how content of a web page should look, where:

- A rule typically comprises:
  - a **selector**, and
  - a **set of properties and values** for how HTML content should be presented

- There are selectors for:
  - **Attributes** e.g. select on attribute name
  - **Element/s** e.g. select on element type

- CSS can be contained in:
  - An external style sheet (recommended)
  - An internal style sheet (sometimes okay)
  - An inline style (not recommended)

Examples of rules

Example 1

- Select (all) <h1> elements, and
- Set three properties: color, background-color, and border

Example 2

- Select (all) <p> elements
- Set one property: color

```
h1 {
  color: blue;
  background-color: yellow;
  border: 1px solid black;
}

p {
  color: red;
}
```

CSS can be contained in:

- **An external style sheet (recommended)**
- An internal style sheet (sometimes okay)
- An inline style (not recommended)

```
<head>
<meta charset="utf-8">
<title>Blah</title>
<link rel="stylesheet" href="style.css”>
</head>
```

CSS can be contained in:

- An external style sheet (recommended)
- **An internal style sheet (sometimes okay)**
- An inline style (not recommended)

```
<head>
<title>Blah blah</title>
    <style>
      h1 {
        color: blue;
        background-color: yellow;
        border: 1px solid black;
      }

      p {
        color: red;
      }
    </style>
  </head>
```

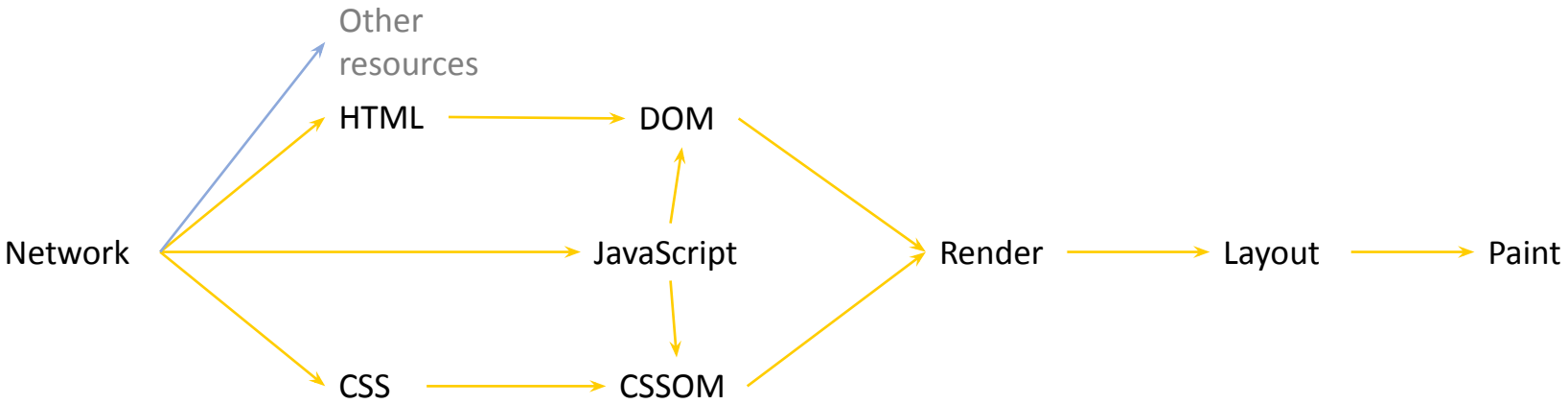
CSS can be contained in:

- An external style sheet (recommended)
- An internal style sheet (sometimes okay)
- **An inline style (not recommended)**

```
<body>

<h1 style="color: blue;background-color: yellow;border: 1px solid black;">Hello World!</h1>

<p style="color:red;">Javascript callbacks are turtles all the way down.</p>

</body>
```

How the client-side technologies fit together

JavaScript, HTML, CSS, DOM...



Advice

Put CSS at the top in the HTML HEAD

Put JavaScript at the bottom of the page

HTML, CSS and JavaScript

- HTML has elements
  - Many pre-defined
  - Define your own:
    - custom attributes
- Elements can be referenced by
  - Element type e.g. `<p>`
  - Unique identifier e.g. `id="…"`
  - Attribute class e.g. `class="…"`
  - (Other ways…?)
- CSS has rules that
  - 'Apply' presentation to referenced elements, through selectors
- JavaScript
  - gets (and sets) …
    - Element content
    - Element attributes & values
  - … based on references to those elements
- HTML document is the primary source
  - Contains HTML (duh)
  - Contains CSS or reference to CSS
  - Contains JS or reference to JS

Content vs data

- **Data** as what is ‘in’ JavaScript data structures e.g. arrays, objects.
  - And therefore what is in your database too
- **Content** as what is ‘in’ the HTML elements.
- Data needs to be **‘injected’** into HTML content e.g. JavaScript setter
- User entered information needs to be **retrieved** from the rendered fields on screen e.g. JavaScript ‘getters’

Looking ahead:

- **Frameworks** help us do this through **two-way binding**.
  - ‘Injecting’ data into content; and retrieving content into data