UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture: Week 6: Security and Introduction to Web Clients

Ben Adams
Course Coordinator
benjamin.adams@canterbury.ac.nz
310, Erskine Building

Agenda for this week's session

- Primer on Security issues
- Introduction to client-side technologies and concepts
- Assignment 2
- Assignment 1 queries
- Mid term exams will be marked by the end of the week.

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

Open Web Application Security Project (updated 2020)

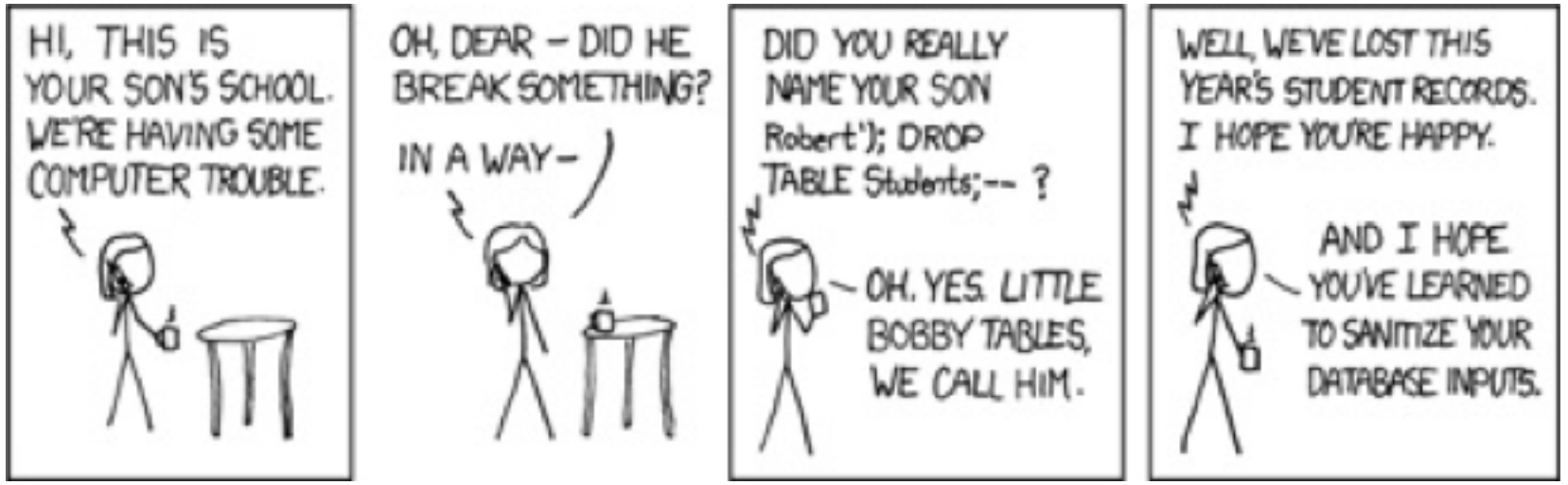
Top 10 security problems
1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging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

https://owasp.org/www-project-top-ten/#OWASP_Top_Ten_Cheat_Sheet

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 e.g.

- `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`

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

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 etc.

- Definitions
  - Authentication: establish claimed identity
  - Authorisation: establish permission to act
  - Authentication *precedes* authorisation
- Why authenticate?
- How can we authenticate?
- Three factors…

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

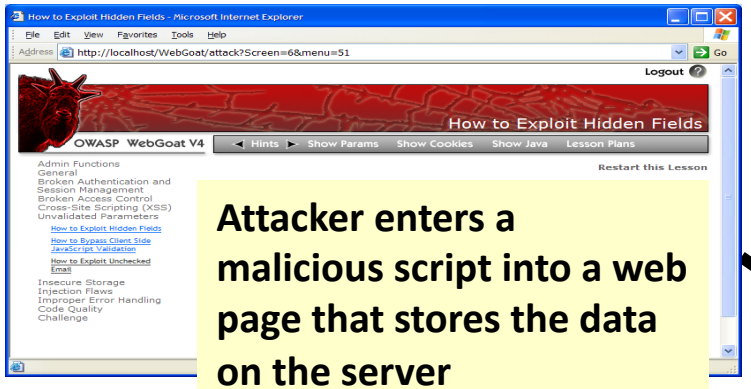
- Forget automated analysis approaches
- Check your SSL certificate
- Examine all the authentication-related functions
- Verify that logoff actually destroys the session
- Use OWASP's WebScarab to test the implementation

Follow the guidance from

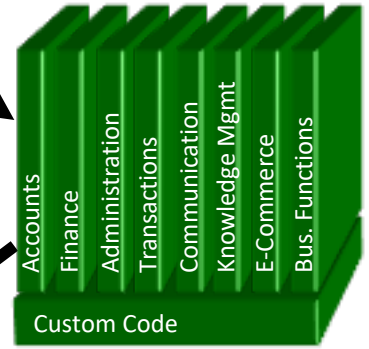
- https://www.owasp.org/index.php/Authentication_Cheat_Sheet

Cross-Site Scripting (XSS)

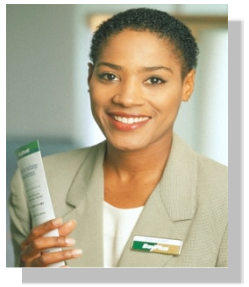
1 **Attacker sets the trap – update my profile**

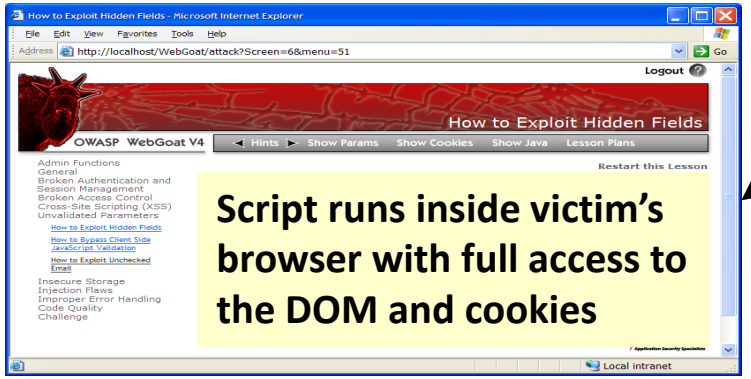


Application with stored XSS vulnerability



2 **Victim views page – sees attacker profile**





3 **Script silently sends attacker Victim's session cookie**

https://owasp.org

DOM-based XSS Injection

- DOM Based XSS allows an attacker to use the Document Object Model (DOM) to introduce hostile code into vulnerable client-side JavaScript embedded in many pages.
- Browser interprets .js, HTML, the DOM etc

DOM-based XSS Injection

- DOM based XSS is extremely difficult to mitigate against because of its large attack surface and lack of standardization across browsers.

- 1. Untrusted data should only be treated as displayable text. Never treat untrusted data as code or markup within JavaScript code.
- 2. Always JavaScript encode and delimit untrusted data as quoted strings when entering the application (Jim Manico and Robert Hansen)

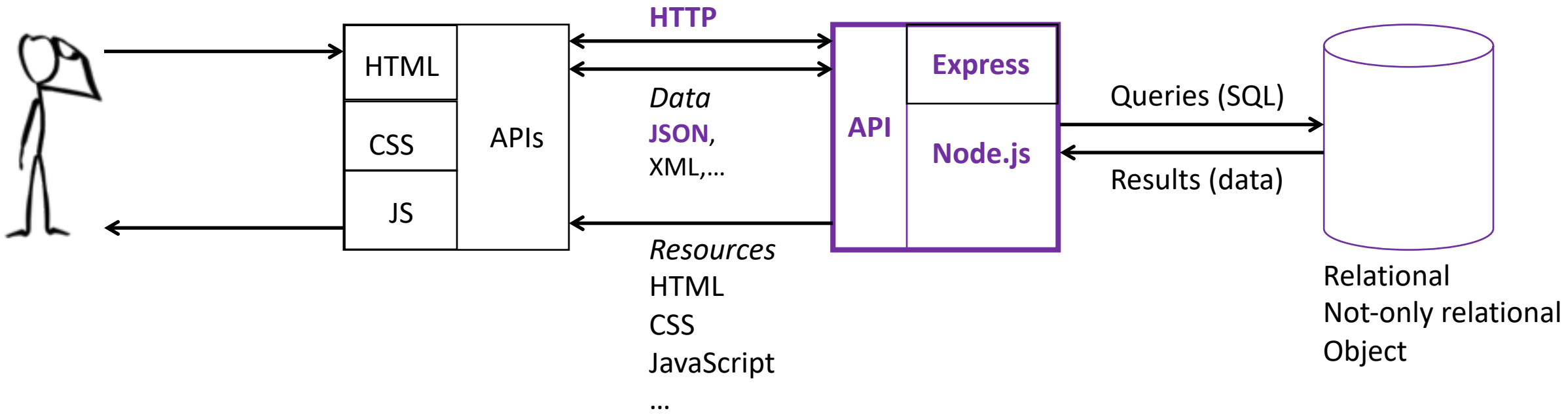
Securing HTTP

- HTTP is a stateless protocol
  - (yeah, we know, you keep telling us…)
- **Each** request contains all the information needed for the server to service that request
  - Remember: GET, POST, PUT, DELETE etc.
- Authentication precedes authorisation
  - Authenticate: establish identity
  - Authorise: grant permissions to act
- **Each request** needs authenticating before authorising
  - e.g. establish identify before serving up a resource

Other approaches (not prioritised)

- 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)

Term 1:
What happens on the server side?



User	Client	Server	Database
Human	Machine	Machine	Machine

What happens on the 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

- JavaScript
- Node.js + express
- Data persistence
- APIs

Additional Lab, Tutorial

- GraphQL
- OWASP Node Goat Tutorial

Term 2:
What happens on the client side?

What is the user experience in the browser?

What content is shown to the user in the browser?

https://www.stuff.co.nz/

What happens on the client side? Rather a lot, actually.

Topics (*indicative*)

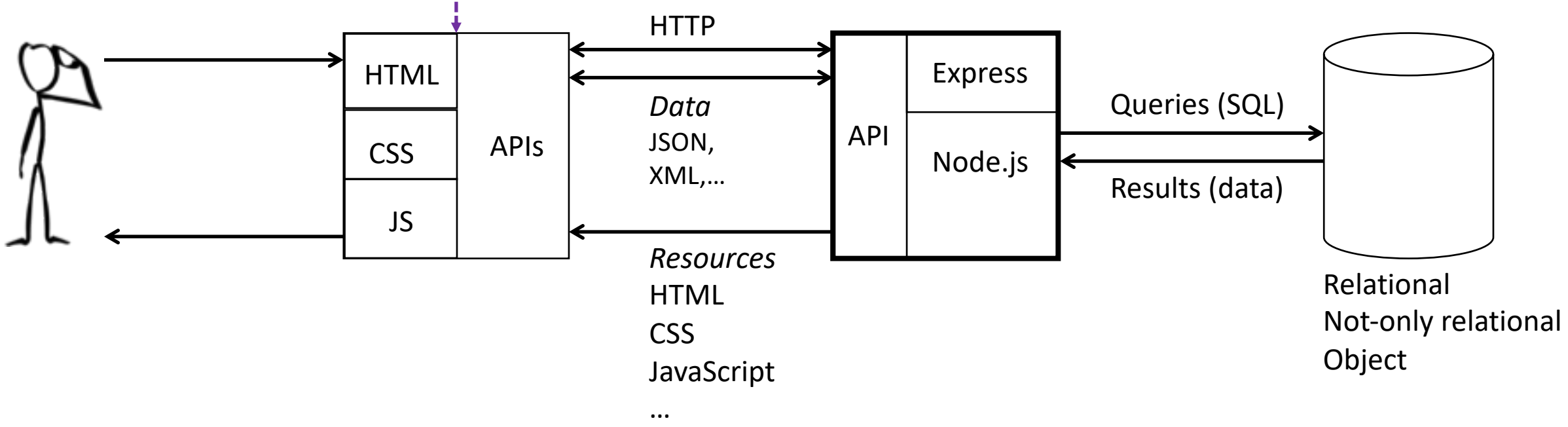
- Web client technologies
  - **HTML, CSS, JavaScript in browser**
  - Libraries e.g. jQuery, Bootstrap,…
  - Frameworks e.g. **Vue.js**
  - **DOM and virtual DOM**
- **AJAX, CORS, …**
- **Design patterns** e.g. MVVM, MVC
- **Testing of web clients**
- (Time permitting
  - Single Page Applications
  - Progressive Web Applications
  - Reactive Programming
  - Web sockets)

Labs

To complement the lectures

- Introduction to client-side scripting
- Vue.js

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:
  - You don't know which browser, or version.
  - You don't know the network-connection quality.
  - You don't know whether cookies are enabled.
  - You don't know 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)

Notes

- 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**.

(There are new elements in HTML5.)

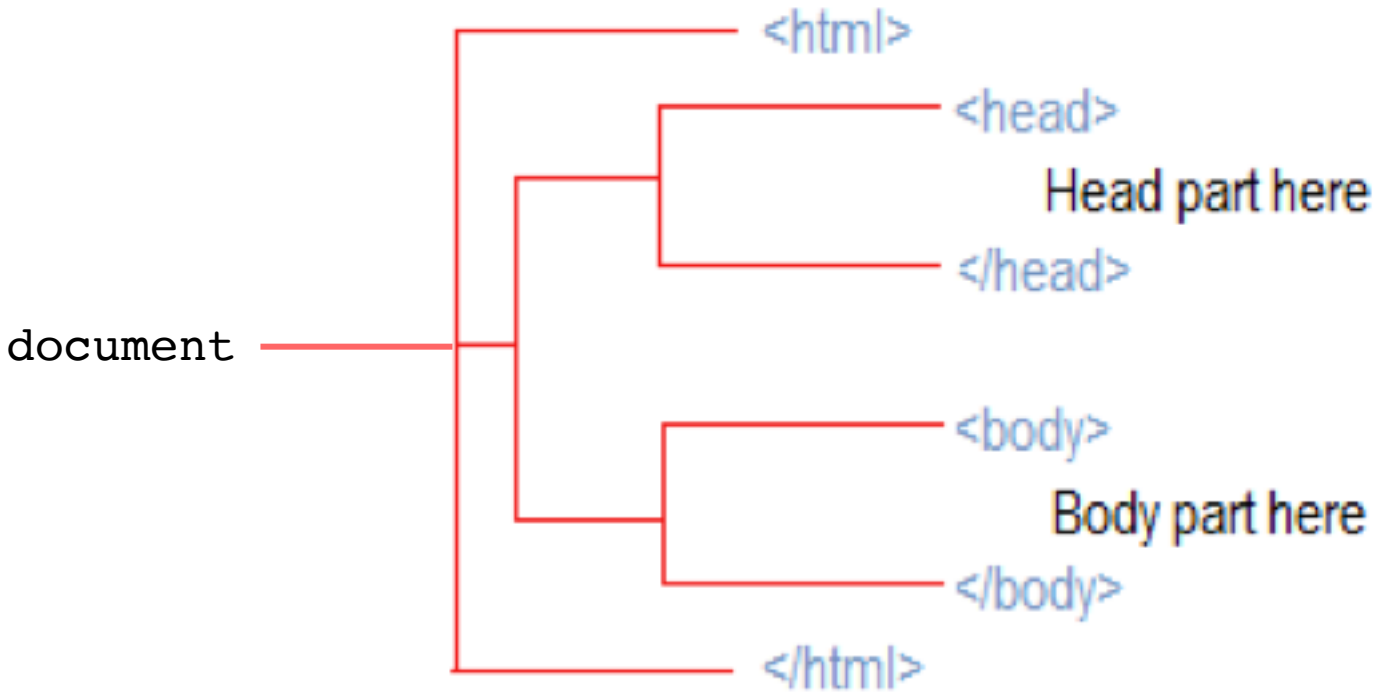
Basic example of HTML page

HTML comprises:

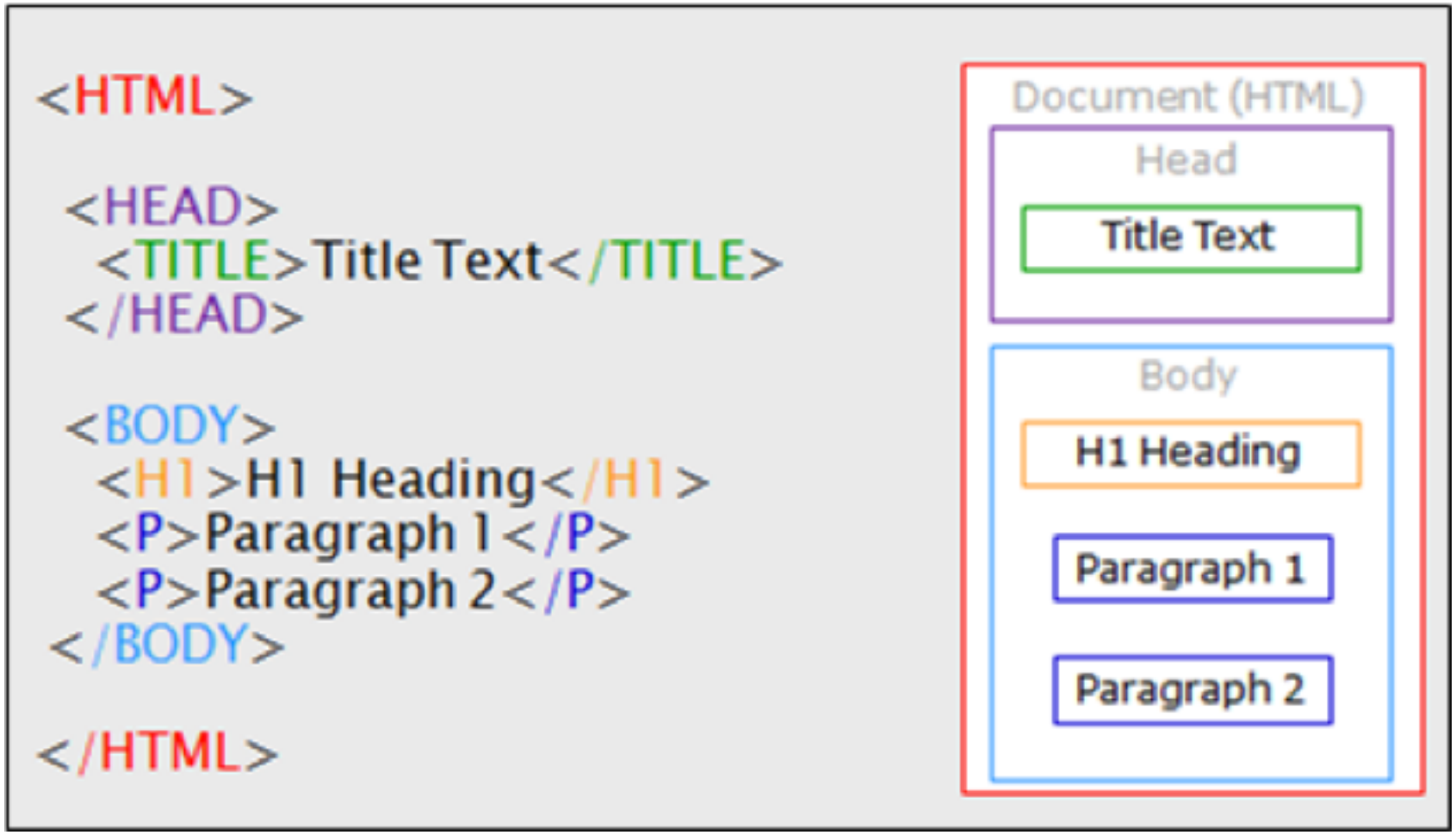
- Document type declaration
- Elements, organised into a hierarchy, e.g.
  - <html>
    - <head>
    - <body>
- Attributes of elements, with values
  - `lang="en"`

```
<!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



http://help.madcapsoftware.com/flare2017r2/Content/Resources/Images/Flare/page_structure_example.png

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

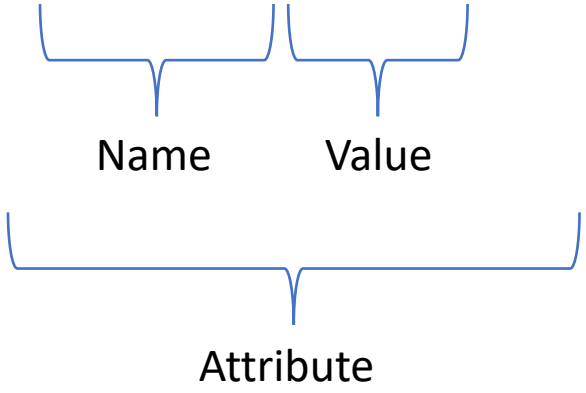
Element

HTML elements & their attributes

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

Javascript callbacks are turtles all the way down.

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



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

Opening tag

Content

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
- Vue.js and Angular.js 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 new 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.

Yeah, but 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 (are you insane?!)

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 (are you insane?!)

```
<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 (are you insane?!)

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

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

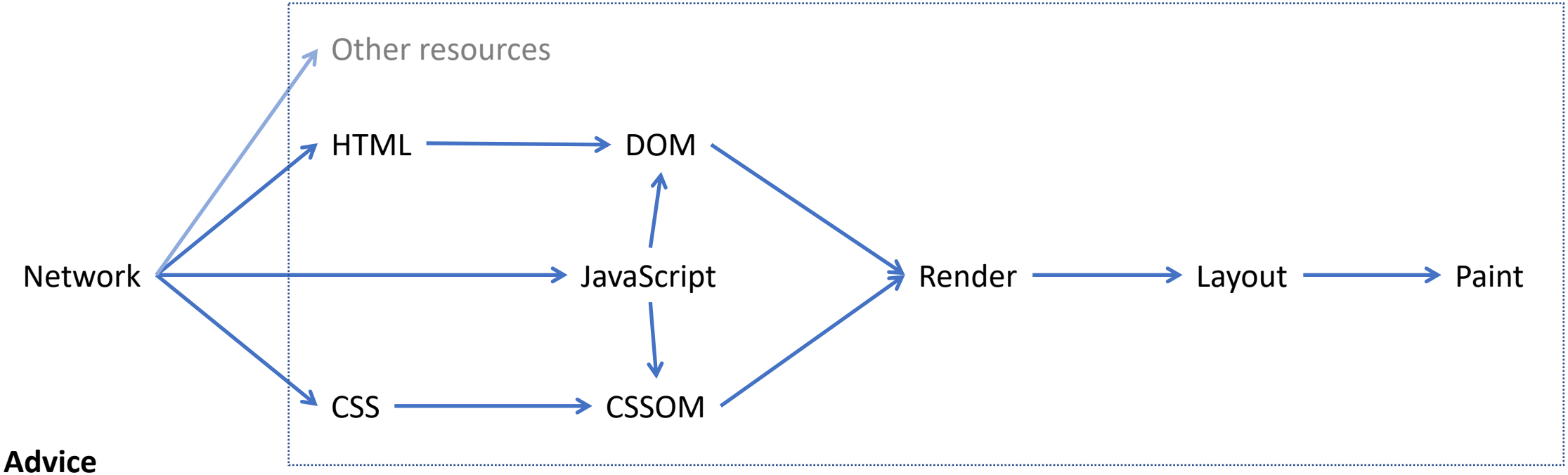
CSS can be contained in:

- An external style sheet (recommended)
- An internal style sheet (sometimes okay)
- **An inline style (Are. You. ‘Insane’?!)**

```
<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

https://developer.yahoo.com/performance/rules.html
https://developer.yahoo.com/blogs/ydn/high-performance-sites-importance-front-end-performance-7160.html

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

Think of:

- **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:

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

What is a webpage?

- Once upon a time:
  - A valid HTTP request resulted in the download of a webpage comprising:
    - Static HTML, CSS, (not even JavaScript) etc, where
    - Data was embedded as the content of the HTML
    - You want new data: request a new page.
- But now:
  - A server may send:
    - Resources: HTML, CSS, JavaScript etc; and separately
    - Data: JSON, XML; and
    - The application on the client-side injects the data into the HTML etc when needed
- Data injection and data retrieval can take place on the client side…
  - Data injected into HTML, and retrieved from HTML
- But still need to be persisted somewhere: locally, or on server

The move toward Single Page Applications

- Users want responsiveness & interactivity; improved user experience
  - Compare user experience with native apps & stand alone apps
- Managing the interactions with a user is *much* more complex than managing communication with server
  - Think of all those events (e.g. `onClicks`) to handle

Single Page Applications (SPAs)

“Single page apps are distinguished by their ability to redraw any part of the UI without requiring a server roundtrip to retrieve HTML. This is achieved by separating the data from the presentation of data by having a model layer that handles data and a view layer that reads from the models.”

http://singlepageappbook.com/goal.html

Some features of Single Page Applications

Separate:

- Data
- ‘Content’ & Presentation: HTML & CSS (and other resources)

Reduce communication with the server/s by:

- Occasional download of resources e.g. HTML, CSS and JavaScript
- Asynchronous ‘background’ fetching of data: **AJAX / XHR or web sockets**
- Fetch data only e.g. JSON
- Fetch data from different servers: **CORS**

‘Page’ navigation

- Client-side JavaScript handles the routing instead of the browser itself
  - Managing page history
- Routing within the SPA

Reconceive web application

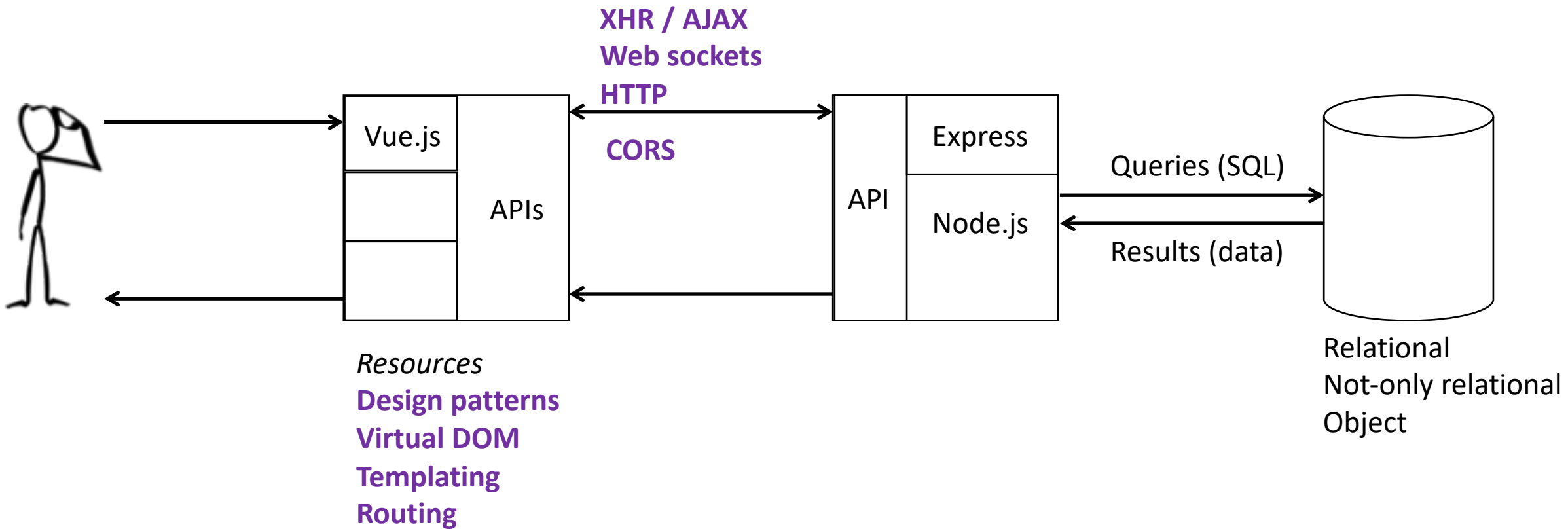
Re-balance workload across:

- Server-side application
  - e.g. Node.js
- Client-side application/
  - Front-end libraries and frameworks e.g. vue.js
- Communication between client & server
  - API-driven/specified
  - e.g. AJAX & JSON

Need to consider (for example):

- Manage application assets/resources
  - e.g. dependency management
- Application design and implementation
  - Modularisation
  - Design patterns
- **Templating**

The balance of work between client and server changes. And there are new technologies!



Labs

Term 2 (6 weeks)

- Week 7: Lab 4
- Week 8: Lab 5
- Week 9: Lab 6
- Week 10 & 11: assignment support
- Week 12: **compulsory** lab
  - Assignment 2 testing

- Will be posted toward the end of the term break.

Assessment

The assessment

- Assignment 1 (25%)
  - **No extension due tomorrow**
- Mid-semester test (20%)
  - Grades by next week
- Assignment 2 (25%)
  - Compulsory lab in final week
  - **No extension**
- Exam (30%; 2hrs)

Assignment 2: web client

Assignment 2

- Assignment Briefing is up on Learn
- User stories are up on Learn
- Implement User stories
- Reference server from Assignment 1 remains online for you to use.
- Source code for Reference server will (likely) be made available on eng-git in due course.
  - Waiting on completion of Assignment 1 matters.
- You will continue to use your MySQL account
- **Make sure you've done up to lab 6**

- How will this be assessed?
  - Students will exercise each others' anonymized client-side apps
  - This will take place in the labs in the last week of term
  - Example scripts for testing online

Example fragment from 2017's test script

1. Can the application be run?		YES	NO
2. Project views			
Precondition: not logged in, viewing a list of sample projects (you might need to navigate to get to this point)			
Steps	**Expected**	**Pass**	**Fail**
1. Page or scroll to see all the sample projects	Should be able to see at least 12 projects (perhaps after scrolling or paging)		
2. Find a search box, and search for `farm`	Two projects: `Let's Raise the Roof (Farm)!` and `The Farmery` should be shown in the project view		
3. Select project `The Farmery` for detailed view	Extra information for project should be shown including rewards, progress towards goal, backers and pledges		
	Totals		
Comments			

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture: Week 6: Security and Introduction to Web Clients

Ben Adams
Course Coordinator
benjamin.adams@canterbury.ac.nz
310, Erskine Building