http://thatjsdude.com/interview/

https://ashleynolan.co.uk/blog/a-guide-to-front-end-interviews

https://github.com/johnpolacek/Front-end-Developer-Interview-Questions/blob/master/README.md (quite old!)

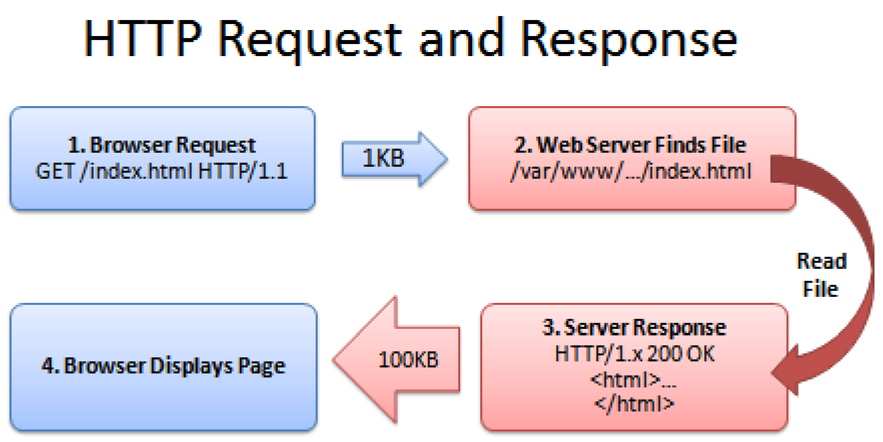
HTML

Doctype – instruction to browser to inform about html version and how to render.

Data-* - allows for data to be stored in DOM which can be accessed later using JS.

Loading CSS/JS – CSS in header as you don't want to restyle HTML content again after it has loaded (prevents FOUC flash of unstyled content). JS in bottom of body as you want all the DOM to load before using.

HTTP Requests



https://www.smartlabsoftware.com/ref/http-status-codes.htm

Accessible Website

Use headings correctly in layout (h1 only for main title and don't skip from h1 to h3 as it can confuse screen reader users.)

Use of semantic HTML tags in order to be more descriptive:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>

Alt text for images.

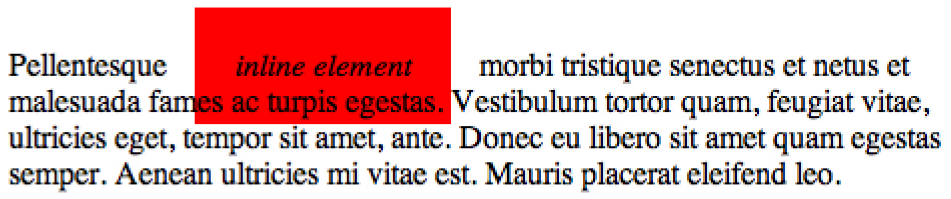
Descriptive and unique link names.

Use colour with care!

Allow for use of keyboard in a logical way.

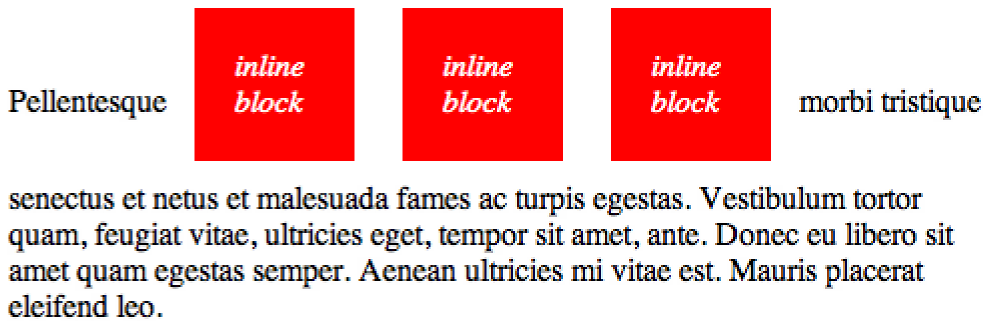
CSS

Display: inline – default value. Don't break flow (like <span>, <em>). Only allows adjoining elements to be spaced horizontally not vertically.



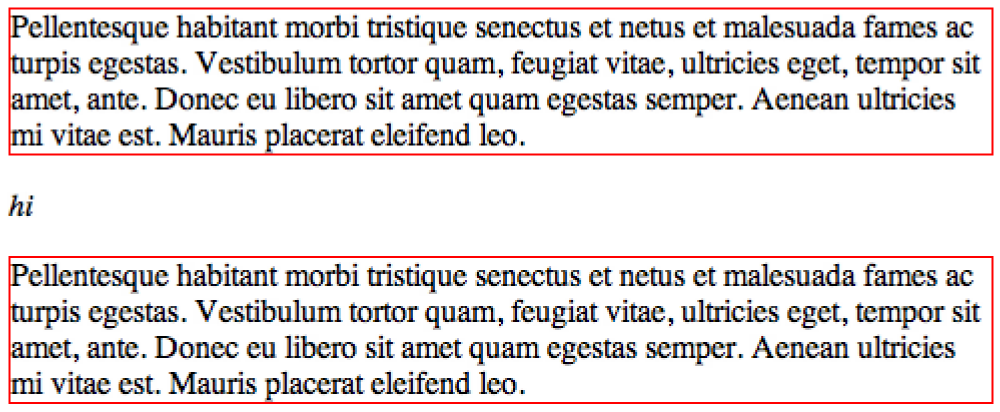
Pellentesque *inline element* morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Display: inline-block – like inline except you can space horizontally and vertically.



Pellentesque *inline block* *inline block* *inline block* morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Display: block – usually container elements (<div>, <section>). Take up as much horizontal space as possible.



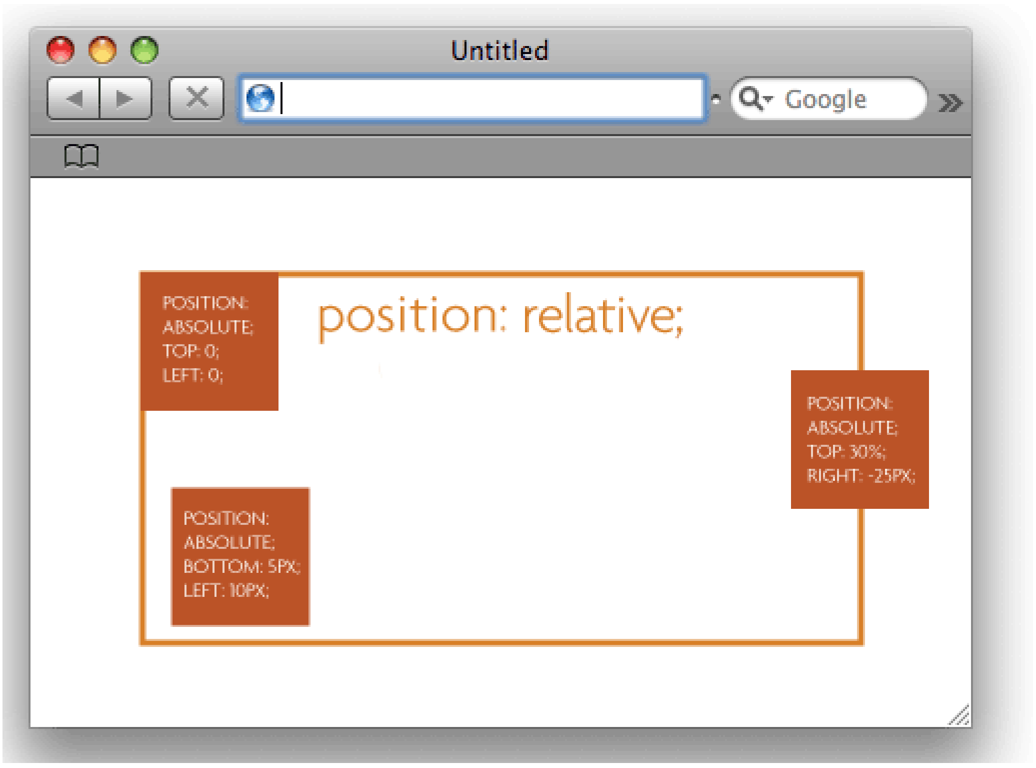
Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

hi

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Position: absolute, relative, fixed –

Absolute position themselves exactly **relative** to their parent element (has to be defined or is body by default).


Untitled
Google
position: relative;
POSITION: ABSOLUTE; TOP: 0; LEFT: 0;
POSITION: ABSOLUTE; TOP: 30%; RIGHT: -25PX;
POSITION: ABSOLUTE; BOTTOM: 5PX; LEFT: 10PX;

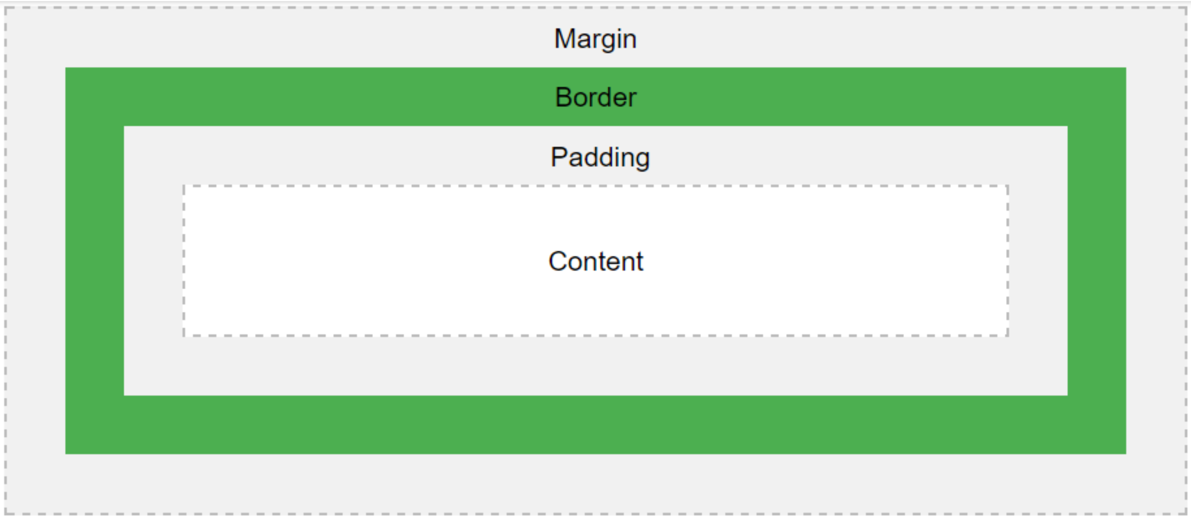
Fixed stays in fixed position on viewport even if scrolled. Good for parallax.

8. box model

Question: What are the properties related to box model?

Answer: Technically, height, width, padding and border are part of box model and margin is related to it.

Extra: Everything in a web page is a box where you can control size, position, background, etc. Each box/ content area is optionally surrounded by padding, border and margin. When you set height and width of an element, you set content height and width.

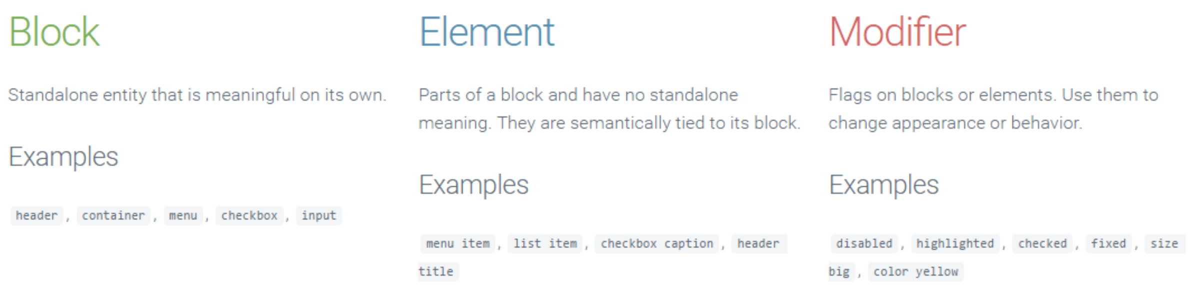


Explanation of the different parts:

- **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

The box model allows us to add a border around elements, and to define space between elements.

BEM



Block

Standalone entity that is meaningful on its own.

Examples

`header` , `container` , `menu` , `checkbox` , `input`

Element

Parts of a block and have no standalone meaning. They are semantically tied to its block.

Examples

`menu item` , `list item` , `checkbox caption` , `header title`

Modifier

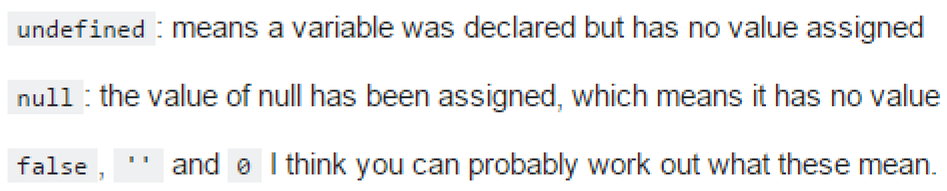
Flags on blocks or elements. Use them to change appearance or behavior.

Examples

`disabled` , `highlighted` , `checked` , `fixed` , `size big` , `color yellow`

<u>JS</u>

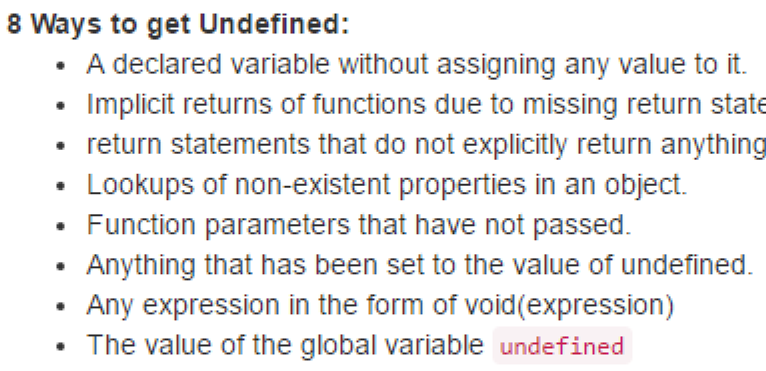
Null and undefined -



`undefined` : means a variable was declared but has no value assigned

`null` : the value of null has been assigned, which means it has no value

`false` , `''` and `0` I think you can probably work out what these mean.



8 Ways to get Undefined:

- A declared variable without assigning any value to it.
- Implicit returns of functions due to missing return statements.
- return statements that do not explicitly return anything.
- Lookups of non-existent properties in an object.
- Function parameters that have not passed.
- Anything that has been set to the value of undefined.
- Any expression in the form of void(expression)
- The value of the global variable `undefined`

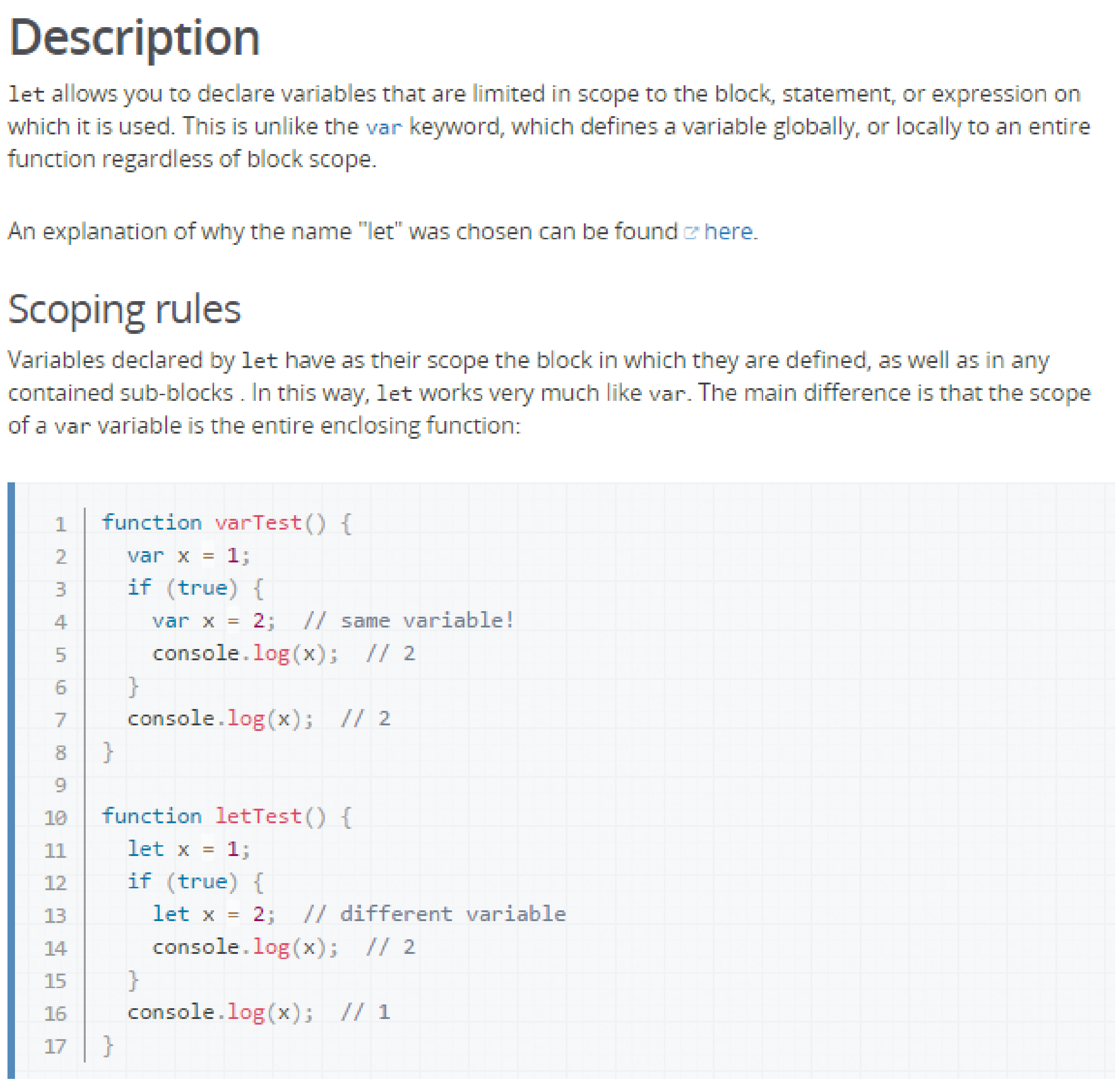
<u>==, == and ===</u>

= assigns value

== checks for same value

=== checks for same value and type

<u>Let keyword</u>



Description

`let` allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the `var` keyword, which defines a variable globally, or locally to an entire function regardless of block scope.

An explanation of why the name "let" was chosen can be found here.

Scoping rules

Variables declared by `let` have as their scope the block in which they are defined, as well as in any contained sub-blocks . In this way, `let` works very much like `var`. The main difference is that the scope of a `var` variable is the entire enclosing function:

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

‘this’ keyword

1. **The *this* keyword always refers to the owner of scope from which it is executing.**

Closures

A "closure" is an expression (typically a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).

The simple explanation of a Closure is that ECMAScript allows inner functions; function definitions and function expressions that are inside the function bodes of other functions. And that those inner functions are allowed access to all of the local variables, parameters and declared inner functions within their outer function(s). A closure is formed when one of those inner functions is made accessible outside of the function in which it was contained, so that it may be executed after the outer function has returned. At which point it still has access to the local variables, parameters and inner function declarations of its outer function. Those local variables, parameter and function declarations (initially) have the values that they had when the outer function returned and may be interacted with by the inner function.

Such as using IIFE in loops.

Call-back function (aka Higher Order Functions)

A higher-order function is a function that can take another function as an argument, or that returns a function as a result.

I have used this in React in terms of .map. In algorithms, .reduce and .filter. Also, have used functions which then callback other functions after relevant parameters are created.

Window.onLoad()

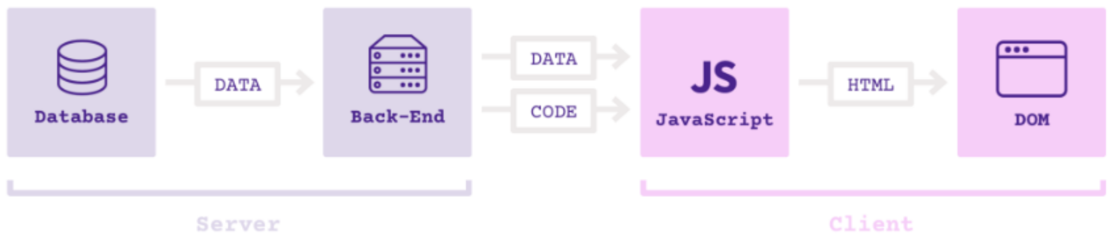
The handler `window.onload` and `iframe.onload` triggers when the page is *fully* loaded with all dependent resources including images and styles.

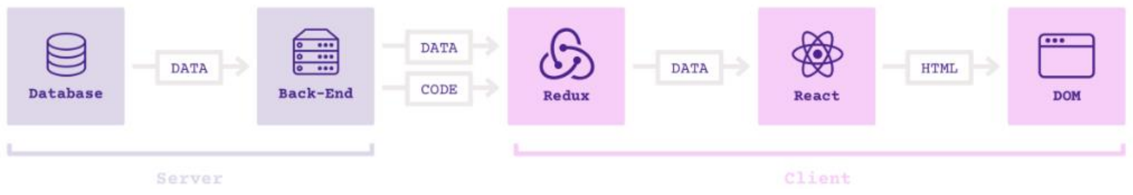
`window.onload` is rarely used, because no one wants to wait until *all* resources load, especially for large pictures.

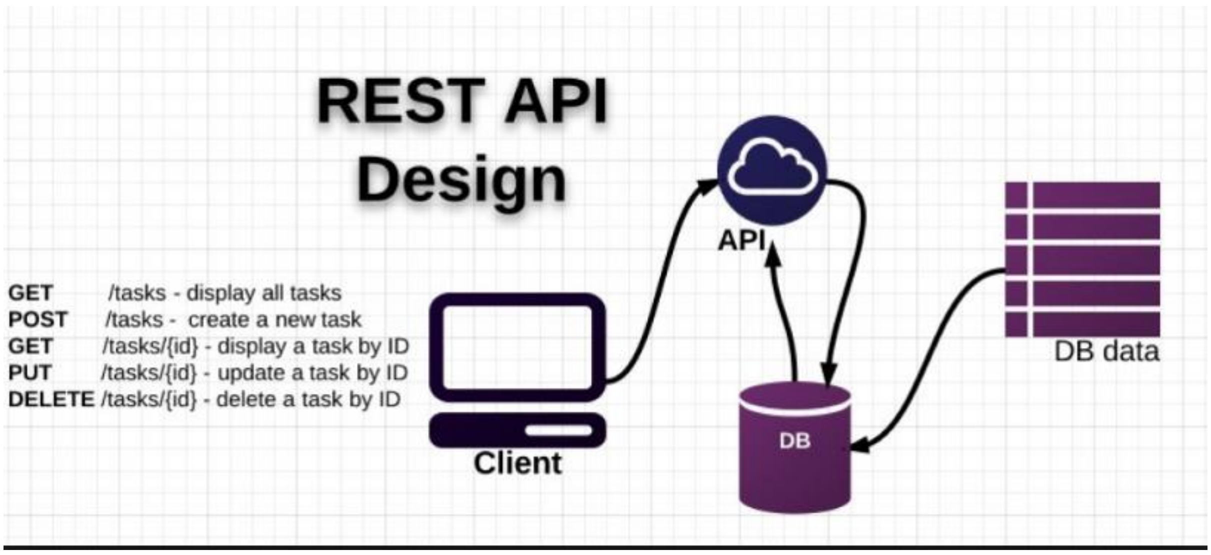
DOMContentLoaded

The `DOMContentLoaded` event triggers on `document` *when the page is ready*. It waits for the full HTML and scripts, and then triggers.

Now compare this with a "modern" 2016 web app (a.k.a. Single Page App):






REST API
Design
GET /tasks - display all tasks
POST /tasks - create a new task
GET /tasks/{id} - display a task by ID
PUT /tasks/{id} - update a task by ID
DELETE /tasks/{id} - delete a task by ID
API
Client
DB
DB data

- Node? What is it? Problems you faced with it? What is Node good for?

- Problems I faced were keeping my Node and NPM version up to date and configuring/sorting any errors related to running on Windows. Also, tempting to use lots of node packages but mindful that need to stick to quality ones, tried and tested with good docs.
- Took JavaScript which normally confined to browser, allowed it to run on computer. Works using Google's V8 engine. Can now access files on computer, listen to network traffic, listen to http requests, can access databases directly (anything you can do with PHP or Ruby-on-rails you can now do with Node JS).
- Allows you to build and use: (a) utilities for everyday development like Gulp/Grunt/Yeoman/Webpack, listen to file changes and hot reload, convert SASS to CSS (b) building a web server using express/koa type framework.
- Modules are basically loading one file into another. NPM downloads and manages packages using package.json which saves all dependencies.
- Uses:
    - I believe Node.js is best suited for real-time applications: online games, collaboration tools, chat rooms, or anything where what one user (or robot? or sensor?) does with the application need to be seen by other users immediately, without a page refresh.
    - If your server side code requires very few CPU cycles. In other world you are doing non-blocking operation and do not have heavy algorithm/Job which consumes lots of CPU cycles.
    - If you are from JavaScript back ground and comfortable in writing Single Threaded code just like client side JS.
    - Not to use for: Your server request is dependent on heavy CPU consuming algorithm/Job.

- What do you like about ES6 and React?

- Modular structure when combined with ES6.
- 1. Your view is composed of components.
  2. Components are rendered using the `props` passed to them from parent components.
  3. A component will be completely re-rendered when the props passed to it change.
  4. Top level components can also maintain `state` to let the re-render themselves and their children dynamically based on user interaction.

- It makes writing JavaScript easier - Being able to drop a bit of HTML in your render function without having to concatenate strings is fantastic, and after a while it feels very natural. React turns those bits of HTML into functions with a special JSXTransformer.
- Components are the future of web development - gives you the ability to create your own components that you can later reuse, combine, and nest to your heart's content. I've found this to be the single-biggest productivity boost because it's so easy to define and manipulate your own components.
- React is extremely efficient - React creates its own virtual DOM where your components actually live. This approach gives you enormous flexibility and amazing gains in performance because React calculates what changes need to be made in the DOM beforehand and updates the DOM tree accordingly. This way, React avoids costly DOM operations and makes updates in a very efficient manner.

- <u>What do you for styling your pages?</u>

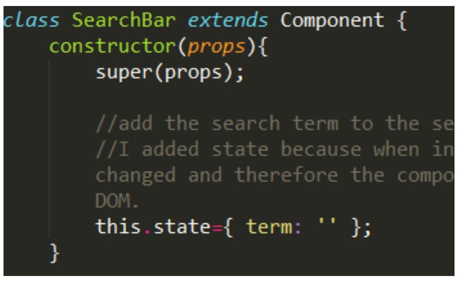
- SASS. Like the ability to embed styles, use mixins (grid, media queries), assign variables.
- Bootstrap for responsive.

<u>- What do you do in able to have websites show on different browsers/devices?</u>

- Autoprefixer as part of PostCSS loader in Webpack (Autoprefixer is a PostCSS plugin). May use CSSnext as it contains autoprefixer and transforms new spec CSS into more compatible CSS.
- Have to test on different devices and use Stack Overflow etc. For example, the Flickr task, I was having issues with the font-weight: bold for the heading as this was causing rendering issues on the iPad. Looked into it and seems it is an issue in using font-weight so took the advice and imported both font-weights of Montserrat font and used bold for the heading and this solved the issue.
- Aware that there are services that allow you to test on multiple browsers but I don't have access. Recently used http://responsiv.eu/.

- Class based component?

- My understanding is JavaScript classes provide a much simpler and clearer syntax to create objects and deal with inheritance.
- The extends keyword is used in class declarations or class expressions to create a class as a child of another class. The super keyword is used to call functions on an object's parent. The constructor initializes this object and can provide access to its private information. The concept of a constructor can be applied to most object-oriented programming languages. Essentially, a constructor in JavaScript is usually declared at the instance of a class.



```
class SearchBar extends Component {
    constructor(props){
        super(props);

        //add the search term to the se
        //I added state because when in
        changed and therefore the compo
        DOM.
        this.state={ term: '' };
    }
```

The 'SearchBar' class is a child of the React component class (extends) and is able to call all the parent functions (super), such as render(), when the class is created (instantiated).

- Webpack? Task runners?

- Module bundler (Webpack) - module bundlers are used to splice various assets and produce bundles to implement a structure into these giant piles of unreadable code.
- Task Runners (Gulp/grunt) - Bundling all your assets into one or even several files and including them to every part of your application means loading a ton of assets that aren't required most of the time. task runners to automate everything that can be automated (i.e. compile css/sass, optimize images, make a bundle and minify/transpile it)
- Simply put, Webpack is such a powerful tool that it can already perform the vast majority of the tasks you'd otherwise do through a task runner. For instance, Webpack already provides options for minification and sourcemaps for your bundle. In addition, Webpack can be run as middleware through a custom server called webpack-dev-server, which supports both live reloading and hot reloading (we'll talk about these features later). By using loaders, you can also add ES6 to ES5 transpilation, and CSS pre- and post-processors. That really just leaves unit tests and linting as major tasks that Webpack can't handle independently. Given that we've cut down at least half a dozen potential gulp tasks down to two, many devs opt to instead use NPM scripts directly, as this avoids the overhead of adding Gulp to the project (which we'll also talk about later).
- The major drawback to using Webpack is that it is rather difficult to configure, which is unattractive if you're trying to quickly get a project up and running.

- Post CSS?

- PostCSS is a tool for transforming styles with JS plugins. Such as Autoprefixer.

- MongoDB? SQL/NoSQL? Express?

- Unit Testing/TDD/BDD?

- Unit Testing gives you the what. Test-Driven Development gives you the when. Behavior Driven-Development gives you the how.
- A unit test focuses on a single "unit of code" – usually a function in an object or module. Individual tests, which test one thing, and they are isolated from each other.
- TDD or Test-Driven Development is a process for when you write and run your tests. Test-coverage refers to the percentage of your code that is tested automatically, so a higher number is better. TDD also reduces the likelihood of having bugs in your tests, which can otherwise be difficult to track down. Start by writing a test
  - Run the test and any other tests. At this point, your newly added test should fail. If it doesn't fail here, it might not be testing the right thing and thus has a bug in it.
  - Write the minimum amount of code required to make the test pass
  - Run the tests to check the new test passes
  - Optionally refactor your code
  - Repeat from 1
- When applied to automated testing, BDD is a set of best practices for writing great tests. BDD can, and should be, used together with TDD and unit testing methods. One of the key things BDD addresses is implementation detail in unit tests. A common problem with poor unit tests is they rely too much on how the tested function is implemented. This means if you update the function, even without changing the inputs and outputs, you must also update the test. This is a problem because it makes doing changes tedious. Behavior-Driven Development addresses this problem by showing you how to test. You should not test implementation, but instead behavior.

- The difference between BDD and TDD is that BDD begins with a B and TDD begins with a T. But seriously, the gotcha with TDD is that too many developers focused on the "How" when writing their unit tests, so they ended up with very brittle tests that did nothing more than confirm that the system does what it does. BDD provides a new vocabulary and thus focus for writing a unit test. Basically it is a feature driven approach to TDD. BDD is TDD done right.
- Mocha is the library that allows us to run tests, and Chai contains some helpful functions that we'll use to verify our test results.

- <u>CORS? Same origin policy?</u>

- From my understanding, To make an AJAX request using CORS, the server needs to be configured to accept cross-origin requests. From my research, Flickr does not support CORS headers on api.flickr.com, so fetch and XMLHttpRequest will not work.
- Same origin policy - is a security concept for the web. Sounds sophisticated, but only makes sure a web browser permits scripts, contained in a web page to access data on another web page, but only if both web pages have the same origin. In other words, requests for data must come from the same scheme, hostname, and port. If http://player.example tries to request data from http://content.example, the request will usually fail.
- CORS (Cross-Origin Resource Sharing) - If you want to store content on a different origin than the one the player requests, there is a solution – CORS. In the context of XMLHttpRequests, it defines a set of headers that allow the browser and server to communicate which requests are permitted/prohibited. It is a recommended standard of the W3C. In practice, for a CORS request, the server only needs to add the following header to its response: Access-Control-Allow-Origin: *
- Used jsonp by adding '?jsoncallback=?' to the url. This tell jQuery to make a JSONP request in which it creates a new script which embeds Flickr JSON data in a Javascript function call (hence the padding). Since the data will be called via the 'script' tag, it is no longer in violation of same-origin policy. JSONP is convenient and served a potentially business purpose, although it has no built in security and this has to be manually enforced by the developer, which is where possible security issues occur.

<u>- what is Lodash?</u>

- Lodash is a toolkit of JavaScript functions that provides clean, performant methods for manipulating objects and collections. It is a "fork" of the Underscore library.

<u>- Bash?</u>

- Console emulator (Cmder). Able to use all the bash commands as on OSx.