**about me n Intro**

**agile waterfall**

**Definition of Done Overview:**

In software, this may a Definition of Done may be: “Done means coded to standards, reviewed, implemented with unit [Test-Driven Development (TDD)](https://www.scruminc.com/scrumlab-stub/), tested with 100 percent test automation, integrated and documented.”

In a services context, it might look something like this: "Done means every task under the User Story has been completed and any work created is attached to the User Story so the Product Owner can review it and make sure it meets his or her expectations."

**projects brief**

**Webpack - package bundler**

=> css preprocessor like sass, less, stylus, postcss, js transpiler like typescript, coffeescript, babel, html templating like mustache, jade, sourcemap, tree shaking, building components into factory, bundling into single file, assets optimizing, creating sprites, minification of html,css,js

<https://hackernoon.com/how-to-create-library-in-angular-2-and-publish-to-npm-from-scratch-f2b1272d6266>

https://hackernoon.com/how-to-publish-your-package-on-npm-7fc1f5aae600

**express**

**Node**

Event loop, event driven, I/O operations

Modules formats=> esnext, commonjs, systemjs, require js, exports.module of node,es2015

<https://auth0.com/blog/javascript-module-systems-showdown/>

**Traditional loading**

include a script for each functionality in index.html creates network bottleneck.The other alternative is to load a big .js file containing all your project code, but this results in an unmaintainable scripts that causes problems in scope, size, readability, fragility and monolith files.

**IIFEs** solve scoping issues for large projects. When script files are wrapped by an IIFE, you can safely concatenate or safely combine files without concern of scope collision. This lead to tools like Make, Gulp, Grunt, Broccoli or Brunch

However, anytime you want to change one file you have to rebuild the whole thing. Concatenating makes it trivial to reuse scripts across files and makes build optimizations more difficult to implement. How do you even know what code is being used and which is not?

If you are only using one function from lodash or one date utility from moment.js you are actually adding the entire library and just squishing it together. How do you treeshake the dependencies on your code? Also, lazy loading chunks of code can be hard to achieve at scale and requires a lot of manual work from the developer.

**webpack** runs on Node.js, a JavaScript runtime that can be used in computers and servers outside a browser environment.

Now that JavaScript is not running in a browser, how are Node applications supposed to load new chunks of code? There is no html files and scripts tags that can be added to it.

**CommonJS** came out and introduced require, which allows you to load and use a module in the current file. This solves scope issues out of the box and which code is used becomes clear since we need to import each module that we are going to need.

But there is no browser support for CommonJS. There are no [live bindings](https://medium.com/webpack/the-state-of-javascript-modules-4636d1774358). There are problems with circular references. Sync module resolution loader is slow. While CommonJS was a great solution for Node.js projects, browsers didn't support modules. That's when bundlers and tools like Browserify, RequireJS and SystemJS were created to solve this limitation making it possible to write CommonJS modules that run in a browser.

This is why **webpack** exists. It's a tool that not only let's you bundle your JavaScript applications, supporting both ESM and CommonJS, but can be extended to support all different kind of assets like images, fonts and stylesheets.

webpack cares a lot about performance and it's always adding and improving features like async chunk loading and prefetching to help you deliver the best possible version of your project to the user, always caring about loading times and performance.

<https://webpack.js.org/concepts/why-webpack/>

Npm

Creating npm scripts like npm start to execute where it is executed.

<https://blog.npmjs.org/post/118810260230/building-a-simple-command-line-tool-with-npm>

<https://github.com/linclark/github-pages-deploy>

<https://medium.freecodecamp.org/introduction-to-npm-scripts-1dbb2ae01633>

**Express, MongoDb, Mongoose**

We’ll be using [Express](http://expressjs.com/) for this application as it is the de facto standard for a great majority of Node applications today. Mongoose is an [ORM — Object Relational Mapper](https://en.wikipedia.org/wiki/Object-relational_mapping). The official ORM used for MongoDB to be precise. To break it down, we use an ORM to simplify the transfer of data between our application and the database. It maps the data we have in our app to uphold a set of strict rules set by the database. The body-parser module is just a middleware we use to parse our data sent through HTTP requests.

REST, to better understand the 4 actions we have on our disposal to interact with a database. They are called CRUD. Standing for **C**reate, **R**ead, **U**pdate and **D**elete. Using HTTP requests, we can use the respective action to trigger every of these four CRUD operations.

* **POST** is used to send data to a server — **Create**
* **GET** is used to fetch data from a server — **Read**
* **PUT** is used to send and update data — **Update**
* **DELETE** is used to delete data — **Delete**

**sass mixins n concepts**

**Variables**

Usage: color:$border-color

$text-color: #222 !default;

**Placeholder**

.frst\_para { color: green;}

.sec\_para { @extend .frst\_para; font-size:20px;}

used the **@extend** directive, which allows one selector to inherit styles of another selector

**Mixins**

|  |
| --- |
| @mixin br() { |

|  |
| --- |
| border-radius: 0; |

}

@include in the css like body {@include br()} // used to add common styles or create as shared snippet and used as functions with parameters and used as nesting also and as autoprefixer

**Function**

|  |
| --- |
| @function calculateRem($size) { |

|  |
| --- |
| $remSize: $size / 16px; |

|  |
| --- |
| @return $remSize \* 1rem; |

}

Can use function to manipulate background text colors and calculate between css units px,em,rem

**Conditional loops**

@mixin on-event($self: false){

@if $self{

&:hover{ @content; // this will be given at declaration part of scss }

@else { &:focus{ @content; } }

}

}

**Map set**

|  |
| --- |
| /// Breakpoints map |
|  |

|  |
| --- |
| /// @prop {String} keys - Keys are identifiers mapped to a given length |
|  |

|  |
| --- |
| /// @prop {Map} values - Values are actual breakpoints expressed in pixels |
|  |

|  |
| --- |
| /// @see {mixin} respond-to |
|  |

|  |
| --- |
| // $breakpoints: ( |
|  |

|  |
| --- |
| // 'small': (min-width: 320px), |
|  |

|  |
| --- |
| // 'medium': (min-width: 768px), |
|  |

|  |
| --- |
| // 'large': (min-width: 1024px), |
|  |

// ) !default;