Section 23

What is Node.js?

- About Frameworks
  - Why use Frameworks?
    - Depending on
      - Back-end Programming Languages
        - We can find compatible framework
    - Framework is a bit like mayonnaise
      - To make Mayo Sandwich
        - You need mayonnaise.
          - To create it
            - You need the egg
              - and the vinegar
              - ...
          - You can just buy it in the shop
            - Buy mayonnaise framework
            - it is so commonly used condiment
    - Web-framework provide pre-built components
      - To reduce overhead
      - Examples
        - Writing Files
        - Reading Files
        - Networking
        - URL Strings
        - Data Streams
        - Debugger
        - Tests
        - Diagnostics
        - Error Codes
      - Node.js does not provide pre-built components
  - Node.js itself is not a framework
    - It is a runtime environment for executing JavaScript on the server side
      - *it is designed to build scalable network apps*
    - When JavaScript was first created
      - it's purpose was to run in a browser
      - JavaScript code was locked in the browser land jail
        - **Node.js** liberated the JavaScript
  - Node.js uses the V8 Engine
    - ~~not eight cylinder V8 Engine~~
      - but this from chromium
      - written in C and C++
  - Node.js uses an asynchronous, event-driven architecture
    - Our previous scripts runs from top to bottom
      - one road lane
        - with one car
    - Node.js operates like a system of independent roads
      - with multiple cars
  - Why Node?
    - So we can have JavaScript Everywhere
      - One language to rule them all
      - JS Fullstack
    - Easy Scaling of applications
    - Non-blocking
    - Huge ecosystem npm
      - vibrant comunity
      - a lot of resources
      - We're not the only people choosing it

The Node Package Manager and Installing External Node Modules npm

- What is Node Package Manager?
  - It's a place which collects modules that people have built for Node
  - If we think of the native Node modules as the toolbox that we have at home
    - then the NPM is the place where you'll find all the weird and wonderful tools
- Initialise NPM
  - `npm init`
    - initializing a new project and creating a package.json file
- Install NPM Package
  - `npm install <Something>`
  - `npm i sillyname`

```
var generateName = require('sillyname');
var sillyName = generateName();
console.log(sillyName);
```

```
$ node index.js
Aldermyth Arrow
```

- In a way, it's kind of like playing with Lego
  - there are plenty of lego pieces created by people or the node team
    - you can add them to your project
- ECMAScript Modules
  - CJS ESM
    - CommonJS (CJS)
      - `const fs = require('fs');`
    - ECMAScript Modules (ESM)
      - `import * as fs from 'fs';`
  - Add to package.json

```
"main": "index.js",
"type":"module",
Debug
```

```
import generateName from 'sillyname';
```

[Project] QR Code Generator

- Packages from npm
  - `inquirer.js` — allow us to get inputs from the user
  - `qr-image.js` — generate qr images
- `npm init -y`
  - so you don't need to type everything manually

~ Tip from Angela ~ Step Up to the Challenge

- Look out for these moments where you're doubting yourself

Using Node.js

- Instalation process
  - Verifying successful installation

```
$ node -v
v20.11.0
```

- Node REPL
  - Read-Eval-Print-Loop
    - Interactive playground for quickly testing and running JavaScript code outside a browser.
      - Similiar to JavaScript console in the browser
  - How to start the REPL?
    - `node`
  - `.help`

```
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the REPL
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file
```

  - ctrl + c (twice) - to exit
- Running JS Files
  - `node index.js`

How to Use the native Node Modules

- What are Native Modules?
  - They're kind of like your starting tool set
  - Think of these native modules as the pre-bundled games on your system.
- File System Module (fs)
  - This native module allows us to access the local storage
  - To start using it, we need to
    - "import" the code
      - ECMAScript Modules (ESM)
    - "require" the bits of code
      - CommonJS (CJS)
  - `const fs = require('fs');`
    - imports the 'fs' module for file system operations
  - `fs.writeFile(file, data[, options], callback);`
    - this method is used to save data to a file.
    - `'file'`: The name of the file to which you want to write the data.
    - `'data'`: The data to be written to the file.
    - `'[options]'`: Optional settings, such as encoding or mode.
    - `'callback'`: is a function called after the write operation, with an optional error (err) and information about the completion.

```
> fs.writeFile("message2.txt", "Hello from Node251", (err) => {...
  });
```

  - `fs.readFile(path[, options], callback)`
    - this method is used to read data from file.

Presented with xmind