Node JS

run JS not on the browser
front end and back end in same language
☆ brush up ES6 and async JS fundamentals

Browser	Node
DOM	No DOM (server side)
Window	No Window
interactive apps (nav)	Server side
No File System	Filesystem!

Read Eval Print Loop REPL (for playing around)
CLI → running app code in node as .exe

node to get into node Repl in node

open VS code terminal to run node CLI

GLOBALS

in vanilla JS, we have window & DOM b/c no browser
access to globals allow us to get around some of this
→ __dirname __filename
→ will cover require / module soon
→ process → gives info on the environment

Modules

split code into modules and then run app.js
encapsulated code only share minimum

Node uses CommonJS and ✡ every file in node is module ✡

modules.js has modules like includes
export { } what you will ship to other modules

require('./names.js') include names

return val you get variables sent by module
we are in charge of what is shared
module.exports = { var1, var2 }

Function called in a module and then included
will fully run and output by itself

Call back functions

Call back: function passed to another and is executed after some operations are done

```
function handleName(name, callback) {  // cb
    const fullName = `${name} smith`
    callback(fullName)
}

handleName('peter', makeUppercase);  // PETER SMITH
                                     // htims retep
```

no pareths

" " " " reverse String

also

```
handleName("name", function(value) { });
```

Call back Hell: nesting multiple callbacks that make code clunky

event 1 → event 2 → event 3 than

nested setTimeouts

resolve → then
reject → catch

promise.then(cb.catch(cb))

Promises (used to avoid cb hells)

3 states Pending, Rejected, Fulfilled

Typically use Promises w/ async await

Order food, await promise is recept, (http request)
Promise constructor w/ cb functions

functions

```
Const promise = new Promise ( (resolve, reject ) => {
    resolve ('hello')
    reject ('failed')
}
console.log(promise)  //output is 'hello'

promise.then ( (data)=> console.log (data))
                                    call back
      log.catch ((error) =>console.log(error))

// new example

  Const promise =new Promise ((resolve, reject ) => {
      Const rand = Math.random()
      if (rand == = value) {
         resolve("guessed correctly")
      } else {
         reject('wrong')
      }
  })

console.log(promise)
```

Promise Example Colors Synchronous

```
const btn = $(document).(.'btn')


('$.btn').click( () => {
   addColor (1000, '.first', 'red')
     .then( () => addColor(3000, '.second', 'blue'))
     .then( () => addColor(2000, '.third', 'green'))
     .catch( (err) => console.log (err))
})      // reject('No valid') ↑


function addColor(time, selector, color) {
   const elem = $(selector)
   return new Promise ((resolve, reject) => {
     if (elem) {
       setTimeout ( () => {        ← call back
         elem.color = color
         resolve(data) // w/o this, promise is Pending
       }, time)                    instead of resolved
     }else {
       reject('No valid elementer')
     }
   })
}
```

---

chain .then when returning promise obj.
Always have a resolve
Can pass data from resolve to subsequent .then()

Async / Await

★ write Async code in synchronous fashion ★
await always waits until promise is settled
can only use await if you have async w/ function
★ Async ALWAYS returns a promise

```
const example = async() => {
    return "hello"
}

async function fun() {
    const result = await example()
    log(result)
}

//console.log(example())
fun()                    // logs → "hello"
```

async always returns a promise

await always awaits for Promise to settle and then get result

Code executes only after await resolve

Async = Makes a function returns a promise
await = Makes an async function wait for a promise

replace .then()

await return (Promise)

Async example

http requests are typically async

Only after getUser has been executed (resolved) .then then getArticles can be called
↑ return promise

try/catch block

Fetch API

performs a get request and reads from dB and returns a promise

fetch(url) → returns promise from URL

then approach

```
fetch(url).then((response) => resp.json())
          .then((data) => log(data))
          .catch((err) => log(err))
```

---

async

```
const getTours = async () => {
  try{
    const resp = await fetch(url)
    const data = await resp.json()
    return data
  } catch(error)
    log(error)
};

log(getTours().then())
```

Built-in Modules OS | PATH | FS | HTTP

OS → talks to OS & server
var os = require('os') built in module
lots of built in methods / use as needed

---

require('path')
returns platform specific seperator
get filepath path.join('/folder/', 'file');
const base = path.basename(filepath)
get absolute path path.resolve()
helps w/ path/machine independence

Built in Modules

FS' filesystem
async - non blocking sync - blocking
{thing you are grabbing} = require('fs')

sync How to read from file system
readFile("./relative file path", "encoding")
writeFile("./newfile.txt", "text", {append})

---

async readFile, writeFile
* provide a callback when read is done

readFile('./folder/file.txt', function);

Callback hell async pattern

---

sync vs async → threading

Use Async await!

HTTP very brief intro
all will revolve around http module from here on out

Server you are setting up listens at Port 5000
want Server to be constantly listening

nodemon --delay 2.5 app.js

npm node package manager

1) reuse our own code in future projects
2) use code written by other devs
3) share our own solutions

folder containing js code package, modules, dependencies

local dependencies → just for project
npm i <package>

global dependencies → ∀ projects
npm install -g < >

npm init -y → init package.json
json file that has all packages for project

creates node_modules showing modules used
package.json shows which dependencies are used

install externals first

ARROW FUNCTION

const name = (param) => { }

npm init -y

Package.json npm -g install npm

Why package.json is so useful

Configure .gitignore to ignore modules

↳ did not push node_modules so

if package.json exists, it automatically load depend

npm install

install depend to not have to type node

(dev) dependeneves

-D

npm i nodemon ~~--save-dev~~

used while creating app

Scripts "start" : "node app.js"

npm start == node app.js

npm run dev → starts w/ node mon

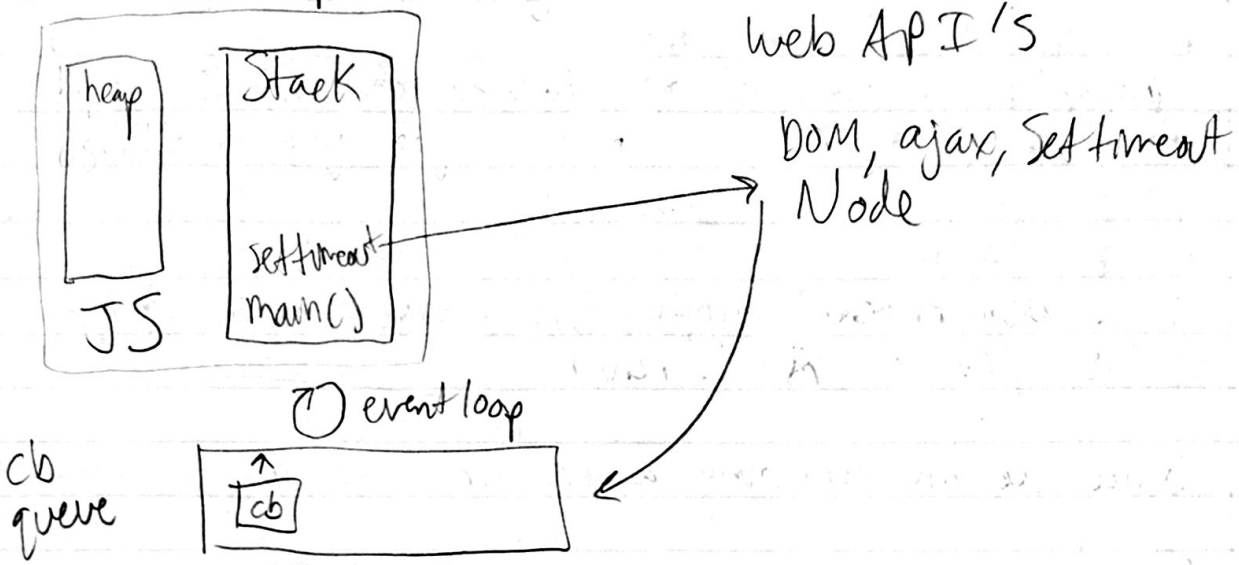
kinda like a live server

npm uninstall

npm install can reload all node_modules

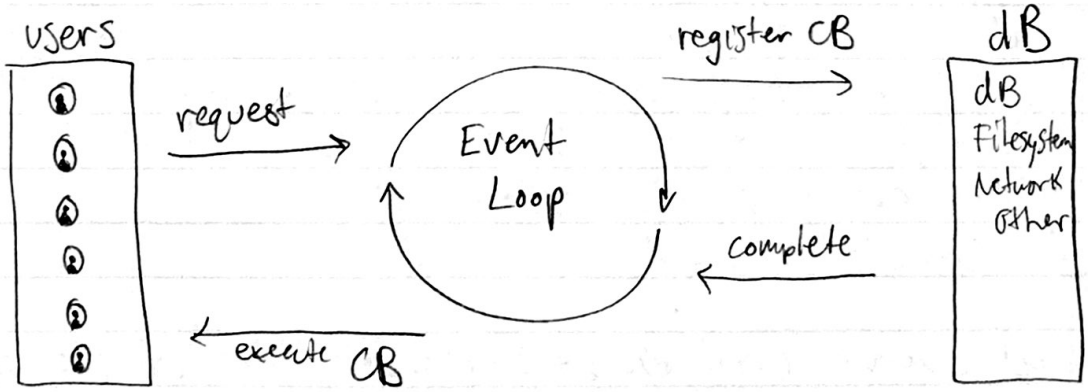
package lock locks versions of when projects were written

Event loop



Event loop is what allows Node.js to perform <u>non-blocking</u> I/O operations despite JS being <u>single threaded</u> by offloading operations to the system kernel whenever possible

JS synchronous means line by line
SetTimeout 0 sec offloads to the browser (API)



Distributing waits, completed task <u>Queues</u> cb

Event Loop Examples alt + 1

node app.js npm start
async block can block other users
so use await

A process in one dir can block whole server
if ya wish to read multiple files. the write
* use promise * to learn async await

Since we are returning a promise, use async await

Set up code w/ wrapper UTIL
require("fs").promises;

Native Options ↑

EVENT emitters

event driven programming is very important to Node

on -> listen for emit
emit -> sends event

Several event call backs can occur after an emit.

.on(data, callback)
↑
event name