Before starting, you must:

* Install Latest version of NodeJS (nodejs.org)
* Install Visual Studio Code.

Then, you can just jump to section 2, part 5.

There is no window/document object. There is global/process.

One of the most fundamental features of Node: Module System

To see the convention name of a module, and it’s name that we can use in require, check Docs.

Git repo for the course: <https://links.mead.io/nodecourse>

Module.exports = name; *(in a file.js we gonna require in some other file.js for the name variable)*

To install a module globally, we must add the -g

To install it with admin rights(?), use “sudo” prefix. (sudo npm I [nodemon@1.18.5](mailto:nodemon@1.18.5) -g)

Nodemon app.js => will update the file at any save done to app.js

To terminate the Nodemon, press CTRL+C

Process.argv => argv = arguments variable

Package: yargs => will parse the arguments

A screenshot of a computer

Description automatically generated with medium confidence

^ console.log( yargs.argv );

YARGS implements a –help on the file too. (which will show up the commands tied to the file)

The builder property will contain only the options we want as a given command to support.

(options as arguments)

Title can be an option to our function. By default, we are not demanded to mention it.

To enable that, we must use the demandOption: true inside the command builder option.

Text

Description automatically generated

To enforce a type for the demanded option, we can use property type: ‘string’ like.

yargs.parse() => goes through the process of parsing the arguments with all the config details provided

fs.readFileSync() => returns a data buffer (binary code). We can .toString() to get the string.

For debugging, we can just console.log() everything and everywhere.

There is also the Node Debugger that comes with the V8 JS Engine.

* Just add “debugger” somewhere in the script, and run cmd

**node inspect** app.js add --title="Shopping" --body="sst"

* Go to chrome://inspect and under the Target, click on Inspect

Graphical user interface, text

Description automatically generated

When we finished all steps (by pressing the PLAY blue button), we can restart it by using “restart” in the <debug> command.

**Call Stack** => track the execution of our program, by keeping track of all **functions** that are currently running. (like when we get an error, where every function called is listed until the error spotted)

* **Data structure:** add on the top of the list / remove the top item on the list

SetTimeout is not a JS function, and V8 has no implementation for it. Is NodeJS which creates an implementation of setTimeout using c++ and provides it to your NodeJS scripts to use. => it is an async way to wait a specified amount of time, and then have a function run.

So when we call setTimeout, is registering an event with NodeJS APIs, and that is an event callback pair, where the event in this case is simply to wait 2 seconds, and the callback is the function to run.

Another event callback pair might be to wait for a database request to complete, then run the callback that does something with the data.

**Node APIs** => ^

**JS is a single threaded programming language**. You can do one thing at a time.

But that doesn’t mean Node.js is completely single threaded. The code you run is indeed still single threaded, **but Node uses other threads in C++ behind the scenes to manage your events.**

**Callback Queue** => maintain a list of all the callback functions that are ready to get executed.

This is the moment the **Event Loop** comes into play. It looks at the Call Stack, and at the Callback Queue.

If the call stack is empty, it’s going to run items from the callback queue.

npm init -y => answers “yes” to all questions

Use postman-request package instead of request package, as it is now deprecated.

The “postman-request” will always return an object: body {} so we go in response.body.

encodeURIComponent(string input) - to convert the input to a safe string. (will convert ? to %3F for e.g.)

**QUESTION: In our weather API, in the accessed object, there is array weather\_descriptions.**

**When we are destructuring the object, how can we specify an index of that array?**

**If we try { weather\_descriptions[0]:description } it fails: SyntaxError: Unexpected token '['**

HTTP ‘n’ HTTPS modules of NodeJS

const request = http.request( url, (response) => {

let data = “” ;

response.on( ‘data’, (chunk) => {

// allows us to register a handler – different events we can call it for: data.

// this event will fire when data comes in

// it can be called once, or more times, so we must save the chunk in a variable until all finished

})

Response.on(‘end’,() => {

// the end event -> when we are done

})

})

request.on( ‘error’, (error) => { // some error handling });

request.end(); // we are just now sending the request.

//-------------------------------------------------------------------------------------------------------------------

**ExpressJS** jumps in.

Fast, unopinionated, minimalist web framework for NodeJS

npm i express@4.16.4

const express = require(‘express’)

const app = express();

app.get(route, ( info about incoming request aka req, res ) => { // let’s us configure what the server should do when someone tries to get the resource of the specific url (maybe we should back HTML, or maybe JSON)

res.send( “text” ); // sending something back to the requester. (can be html or object/array)

})

app.listen(3000, () => { console.log(‘Server is up on port 3000”) }); // starts the server – development port = 3000 (it’s not a standard)

// port 80 => for HTTP based websites

When doing changes to the code, server won’t be updated. We would be needed to close it and restart.

Instead, we can use nodemon which will reload the code at every change!

Node provides an absolute path to the public directory.

\_\_dirname

\_\_filename

NodeJS Path helps.

app.use( express.static(path.join(\_\_dirname,"../public")) )

express.static() => takes the path to the folder we want to serve up

**Relative path:** ./css/style.css

**Absolute path:** /css/style.css

**Template engine:** Handlebars => allows us to do 2 things:

1. Render dynamic documents as opposed to static ones
2. Create code that we can reuse across

But for using with express, we use: HBS package

Dynamic pages will now have the extension .hbs and will be placed in the project directory, under /views

To load the page ^ with express, we must add a route, and use res.render(index) (instead of res.send)

App.set( ‘view engine’, ‘hbs’ )

App.set( ‘views’, viewsPath )

https://expressjs.com/en/4x/api.html#app.set

hbs.registerPartials( partialsPath ); // sets the directory path for partials

To add a partial in another .hbs file, we must use {{>partial file name}}

To make the nodemon listen to other file changes too (instead of just .js files), we must use:

Nodemon src/app.js -e js,hbs

For a route that was not registered, we can display a 404 error: Not found.

To do that, expressJS gives the route \* .

App.get( ‘\*’, (req, res) => {} );

And can be mixed with another path, like: ‘/help/\*’

But those must be added as latest routes.

**req.query** => provides the key/value pairs from the url

**fetch() is not available in NodeJS, as it is part of the browser API.**