Node.js

Created	@February 8, 2023 1:58 PM
Type	Backend
Materials	https://www.youtube.com/playlist?list=PL4cUxeGkcC9jsz4LDYc6kv3ymONOKxwBU https://www.youtube.com/playlist?list=PL4cUxeGkcC9iqqESP8335DA5cRFp8loyp
Reviewed	☐

What is Node.js

- It is a run time environment to run JavaScript outside of the browser
- Is built upon C++ so interprets JavaScript into C++ code
- uses a V8 engine.

How to use it

- first, install node on your device
- then on a terminal type node then you can write any valid JavaScript
- But it is better if you write the JavaScript in a file then run it like `node file_name`

Differences with Plane JavaScript

- The global object is “global” not “window”. ex - `global.timeout()`, but obviously it will understand if we write “timeout()”
- we don’t have a “document” object

Useful features

- `__dirname` - to access the absolute path of the folder we are currently in.
- “__filename” - same but filename included

Modules

To split code into modules we can export the function, class, variable, object... like `module.exports = something`, to export multiple things `module.exports = { something, other thing…}`

Then we can import them from another file like `const {some-name, other-name} = require('./relative-path')`

File system

We can use the “fs” library to access and use the file system.

- `const fs = require('fs')`
- `fs.readFile("relative-path", (err, data) => {})`
- `fs.writeFile("relative-path", 'what you want to write', callback)`
- `fs.mkdir('relative-path' , err => {call back}`
- `fs.unlink('relative-path' , err => {call back}`

Streams

We use it to access data while it is accessing the file.

We get access to small chunks of data continuously

- const fs = require('fs')
- const readStream = fs.createReadStream('relative-path')
- readStream.on('data', (chunk) => {consol.log(chunk.toString())}

We can also use it to write a chunk at a time. so using this feature we can write what we are reading.

But to simplify this use “pipe”

- create readStream and writeStream
- readStream.pipe(writeStream)

Node package manager (npm)

- It is used to install packages, frameworks and libraries on our project or globally
- first, initialize the project with “`npm init`”
- use “`npm i package-name`” to install a package
- use “`-g`” to install them globally
- “`npm install`” alone installs all the packages listed in the dependencies in the package.json file

Creating a server

We create a server so then we can communicate with the browser.

- `const http = require('http');`
- `const server = http.createServer( (req, res) => {console.log(request made)})`
  - `req.url` - returns the route eg - /blogs
  - `req.method` - get, post

  To send html text
  - `res.setHeader('content-type', 'text/html');` then we can write valid htmls like
  - `res.write('<p>hello</p>')`
  - `res.end()` - to end the response

  To send pre written html page
  - `res.setHeader('content-type', 'text/html');`
  - `fs.readFile('relative-path', (err, data) => {res.write(data); res.end();}`

  To send a page depending on the route the user writes
  - create a path variable then assign a path of the file you want to send based on the req.url (we can use switch to do this)
  - then when you read the file provide the path with the path variable

  While sending the res we can send the status code with it like "`res.statusCode = the-code;`"
- `server.listen(port-number, 'host-name(eg- local host', () => { console.log(listening to port …) })`

Express.js

It is a node framework to create node servers easily.

- Install express "`npm i express`"
- import "`const express = require("express")`"
- create express app "`const app = express()`"
- then listen to requests " `app.listen(port-number)` "
- We can use the same route name for different methods like get or post since they are processed differently
- To listen to **get** request(get resources from the server) "`app.get('route', (req, res) => {})`"

  we can use the methods we used before like "write() and end()" but It is easy to use "send()"

- `res.send("a valid html")` - to send html
- `res.sendFile('relative-path', {root: __dirname});` - root is used to tell where to start the relative path.

Redirecting

When a user inputs some route to change it to another route

- `app.get('/route-to-redirected', (req, res) => {res.redirect('/route-where-you-want-to-redirect-to')}`

Sending a 404 page

If the user inputs an invalid route. put it after all the routings.

- `app.use((req, res) => {res.status(404).sendFile('relative-path-of-the-page-to-be-sent', {root: __dirname})})`

- To listen to **post** request(to create a new data or send a data to the server) `app.post('route', (req, res) => {})`

Accepting a form

When you hit enter the data is going to be sent as an name-value pair. and could be accessed with their names.

- `<form action="/the-route" method="POST"></form>` - use this attributes in the form
- `app.use(express.urlencoded())` - to change the form input to a JavaScript object
- `app.post('/the-route', (req, res) => {console.log(res.body)})`

Route parameters

are values that are passed in the URL of a web page to identify a specific resource or page within a website such as the ID of a user, a specific product, or a specific post in a blog. eg https://mypersonal.com/journal/e456

- `app.get('/the-route/:id', (req, res) => {console.log(req.params.id)}` - to access these from the server. we can use whatever we want in place of id
- they are like a variable of routes we use them to dynamically request resources

Middleware

Is any function that run between the request and the response like " use(), get(), post()..."

- eg `app.use(express.json());`

View engines(ejs)

It is an html like engine that enable us to make the html dynamic and make templates and enable us to write a JavaScript in html.

- has a .ejs extention
- first install it `npm i ejs`
- register the view engine in the server - "app.set('view engine', 'ejs')"
- then create files with .ejs extension in the "views" folder
- It is the same as html but when we want to use JavaScript you have to put them in `<%  %>`
- never put the html tags in "<% %>"
- use `<%=  %>` to output the value
- when you want to render that file `res.render("file-name")`

Code Splitting

to avoid repetition in our html we can put some portion of it in a separate file then include it. eg - header, navs...

- <%- include('./relative-path') %> - we use "-" to escape special characters
- usually we put them in the partial folder in views

.env file

- Used to store environment-specific variables
- key-value pair usually set by the operating system
  - type "`gci env:* | sort-object name`" in power shell to see all the env variables of current pc
- we can set our own custom environment variables like API key, eg we can store the API key for a weather app since the the location of the users may vary
- sometimes we use it to store some sensitive information we don't want to be revealed
- doesn't have to be pushed to GitHub or production

how to use it

- create a `.env` file in the root dir

- put the name-value pair you want - eg `API_KEY=2333EIYEH3345533DHIEH`
- Install the dotenv package `npm i dotenv`
- then in your app `require('dotenv').config();`
- then you can use `process.env.the-env-name`

User Authentication

Guide line for a better security

Nodejs_Security_Cheat_Sheet.html