Node.js & Express.js

CHEAT SHEET

Nodejs

Node introduction
Node module System
Node Package Manager

Expressjs

Express js introduction

Building web server with express

Building web server – Basics
GET Requests
POST Requests
Input validation
PUT Requests
DELETE Requests
Working with form data
Rendering css & images
app.js copy-paste template

More express

Middleware
Environments
Configuration
Debugging
Templating engines – pug
Templating engines – EJS

Why node ?

Node.js **What it is ? →**	• It is a runtime environment for executing JS code outside of a browser. • Node apps Are single threaded means a single thread is used to handle multiple requests • Node apps are asynchronous by default • It Is a c++ program that embeds chrome's v8 JS engine
Node.js **Features →**	• Has a large ecosystem of open source libraries & comes in with a lot of built in modules • is used to build backend services. • Is ideal for building highly-scalable, data-intensive & real-time apps • Is great for prototyping & agile development. • Ideal for i/o intensive apps • Should not be used for CPU-intensive apps
Node REPL :- • Read Evaluation Print Loop • node REPL allows you to execute code in byte size chunks	**// To access the node REPL simply write node in terminal** node; **// execute JS file** node file_name.js

Node module system

Global object :- • Node does not have window object rather it has the global object • Only functions defined are added into global object & not variables	global.console.log global.setTimeout var message = " "; console.log(global.message); // undefined
modules :- • In node every file is a module & the variables & functions defined in that file are scoped to that module	**// outputs json object with module properties** console.log(module)
Creating a module :- • module exports are instructions which tell node which functions, objects, variables to export	var url = "url" function log(message) { console.log(message) } **// export the log function** module.exports.log = log; or exports.log = log; **// export the url variable** module.exports.endPoint = url;

Node module system

loading a module :- • to load a module the require( ) function is used	**// export the logger.js module created in the same directory** const Logger = require('./logger')
module wrapper function :- • node does not execute our code directly instead it executes it inside of a function	**// the below function represents the module wrapper function** function(exports, require, module, __filename, __dirname) { // js code }
path module :- • The path module provides utilities for working with file and directory paths. • https://nodejs.org/api/path.html	**// access the path module using** const path = require('path'); **// Eg of some methods** **// outputs a json file with some useful properties about current file** path.parse(__filename);

Node module system

OS module :-

- The os module provides operating system-related utility methods and properties.
- https://nodejs.org/api/os.html

```
// access the os module using
const os = require('os');

// Eg of some methods
// shows the total memory of OS
os.totalmen(  );

// shows the free memory of OS
os.freememory(  );
```

file system module :-

- The fs module provides utilities for working with file and directories
- in fs module every method comes in 2 forms – Sync(Blocking) & Async(non blocking)
- https://nodejs.org/api/fs.html

```
// access the fs module using
const fs = require('fs');

// Eg of some methods
// shows files & folders in the current folder in an array
fs.readdirSync( './ ' );

// copy file1.txt to file2.txt
fs.copyFileSync("file1.txt", "file2.txt")

// The async method also takes a callback function as an argument
fs.readdir( './ ', function(err, files){
  if (err) console.log(err);
  else console.log(files); };
```

Node module system

Events module :-

- The events module provides related utility methods and properties for working with events.
- https://nodejs.org/api/events.html

```
// access the events module & the EventEmitter class using
const EventEmitter = require('events');

// Create instance of EventEmitter class
const emitter = new EventEmitter( ) ;

// Add an event listener
emitter.on('messageLogged', function(){
        console.log('Listener called')
});

// Raise an event
emitter.emit('messageLogged')

// Event arguments can also be passed to get some data
emitter.on('messageLogged', function(arg){
        console.log('Listener called', arg)
});
emitter.emit('messageLogged', {id:1, url: 'url'})
```

Node module system

Events module :-

- **Note :** To raise events in your application you need to create a class that extends EventEmitter which will have all the functionality defined in EventEmitter but additional functionality can also be added & inside the class if you want to raise an event you can use this.emit( )
- Inside another file use the instance of custom defined class you have defined that extends the event emitter

logger.js

```
const EventEmitter = require('events');
// create logger class
class Logger extends EventEmitter {
      log(message) {
         console.log(message);
          // raise event
         this.emit('messageLogged', {id:1, url: 'url'})
       }
}
// export the logger class
module.exports = logger;
```

---

app.js

```
const Logger = require('./logger')
const logger = new Logger();

logger.log('message')
```

Node module system

HTTP module :-

- Provides utilizes for creating web servers or backend applications
- https://nodejs.org/api/http.html

```
// Load the http module
const http = require('http');

// Create a web server
const server = http.createServer((req, res) => {
    if (req.url === '/' {
        res.write("Welcome");
        res.end();
    }

   if (req.url === '/api') {
        res.write(JSON.stringify([1,2,3]));
        res.end();
    }
});

// listen on port 3000
server.listen(3000);
```

Node module system

HTTP module :-

http.get(url, function(response){ })

- It takes a API url and a callback function has a parameter
- The callback function returns a response
- To access the data received from the api we use the response.on(“data”, function(data){}) function

```
// Make a GET request
app.get("/", function(req, res){

 // Make a http GET request to a API endpoint
http.get(url, function(response) {

    // Access the data received from the api
    response.on("data", function(data){

      // log the data (hexadecimal format)
      console.log(data)

      // log the data (JS object format)
      console.log(JSON.parse(data))

      // Access temp value from json
      const temp = WeatherData.main.temp

     // Display the temperature
      res.write("Temperature is " + temp)
      res.send()
    }) }) })
```

Node module system

HTTP module :-

http.request(url, options, function(response){ })

- It takes a API url, options dictionary and a callback function has a parameter
- The callback function returns a response
- To access the data received from the api we use the response.on(“data”, function(data){}) function

- To send data to the api server we store the https.request function in request variable then use request.write()
- The we use request.end() to end our request

```
// Make a POST request
app.post("/", function(req, res){

// Options parameter of https.request
const options = {
      method: “POST”,
      auth: “username”:”API_KEY”
}

 // Make a http POST request to a API endpoint
const request = https.request(url, options, function(response) {

      // Access the data received from the api
      response.on("data", function(data){

         // log the data (JS object format)
         console.log(JSON.parse(data))
}) })
// Use the request variable to send the json data to API servers
request.write(jsonData);

// to specify that we are done with our request
 request.end()
})
```

NPM – Node Package Manager

- NPM is node package manager for external modules - https://www.npmjs.com/
- Every Node app has a package.json file that includes metadata about the application

Install/Uninstall npm package →	**// install npm package** npm install package_name or npm i package_name Eg. npm i underscore **// install specific version of package** npm i package_name@version_no Eg. npm i mongoose@6.4.0 **// Uninstall Specific Package** npm uninstall package_name or npm un package_name
Using node package →	**// using 3rd party node package** var un = require('underscore') un.contains([1,2,3], 2); // true

NPM – Node Package Manager

Create package.json file →	**// create package.json file for your node app** npm init; **// Create package.json file without filling in the details** npm init --yes
Updating local Packages →	**// see list of outdated packages** npm outdated **// update package in minor & patch releases** npm update **// to update packages in latest dependencies download npm-check-updates then use :** ncu –u //to update json file npm i // to update package

NPM – Node Package Manager

Viewing registry info for a package →	**// see all meta data of a package** npm view package_name **// view only particular property from meta data** npm view package_name property_name Eg. npm view mongoose dependencies **// see all the versions of a package released so far** npm view package_name versions Eg. npm view mongoose versions
Package/App Dependencies →	**// Download node app dependencies** npm i **// See the list of the installed dependencies and their version** npm list

NPM – Node Package Manager

Publishing & updating a Package →	**// login in npm first** npm login **// publish your package** npm publish **// Update your package** npm version major/minor/patch Eg. npm version minor
Exclude node_modules folder from git repository → • add .gitignore file & in it write node_modules/	**// initialize git repository** git init **// shows files to add in our git repository** git status **// add files in git** git add **// commit** git commit –m “message”

Express JS & nodemon

Express js → • Express.js adds extra features to NodeJS & helps to organize & structure the code • https://expressjs.com/	**// Install express** npm i –g express
Nodemon → • Nodemon monitors changes in code and automatically restarts your server so that you won't have to do it manually	**// Install nodemon** npm i –g nodemon **// execute js file** nodemon file_name.js

Building web server - Basics

Load express module →	**// Load express module** const express = require("express"); **// store express object in app variable** const app = express();
app.get() • Specifies what should happen when a browser makes a GET request • It takes 2 parameters :- 1. path/url 2. callback func. with request & response parameters • The res.send() will send content specified to page	**// Send "hello" to home route** app.get("/", function (*req*, *res*) { *res*.send(" hello "); }); // **Send HTML file to home route** app.get("/", function (*req*, *res*) { // __dirname shows current file path console.log(__dirname); // Send html file *res*.sendFile(__dirname + "/index.html"); });
app.listen() • Will listen at a specific port for any http request	**// Starts & Listens on port server 3000 for connections** app.listen(3000, () => { console.log("Server started on port 3000"); });

Building web server - Basics

Environment variables

- Variable that is in part of the environment in which the process runs & it's value is set outside of the application

```
// Read the value of environment port
const port = process.env.PORT || 3000;
app.listen(port, () => {
   console.log(`Server started on port ${port}`);
});

// set environment variable on windows in terminal
set PORT=5000

// set environment variable on windows on mac
export PORT=5000
```

Route parameters

```
// add a parameter -  :parameter_name
app.get("/post/:year/:month", (req, res) => {
  // show the year entered in url
  res.send(req.params.year);

  // shows request params json
  res.send(req.params);

  // Reading query string parameters
  res.send(req.query);
});
```

Building web server – GET Requests

Handling HTTP GET Requests :-

```
const courses = [
  { id: 1, name: "course1" },
  { id: 2, name: "course2" },
];

// returns array of courses
app.get("/api/courses", (req, res) => {
  res.send(courses)
});

// return course with given id
app.get("/api/courses/:id", (req, res) => {
  // check if course id exists in array
  const course = courses.find((c) => c.id === parseInt(req.params.id));

  // send message if course not found
  if (!course) res.status(404).send("Not found");

  // send course if found
  res.send(course);
});
```

Building web server – POST Requests

Handling HTTP POST Requests :-

app.post( )

- Postman can be used to make post requests

```
// ADD new course
// enable parsing of json objects
app.use(express.json())

app.post('/api/courses', (req, res) => {

    // validate course
    const {error} = validateCourse(req.body); // eqi to result.error
    if (error) return res.status(400).send(error.details[0].message);

   // read the course object that is in the body of request
   const course = {
      id: courses.length + 1,
      name: req.body.name
   }

   // add new course object in array
   courses.push(course);
   // return object in the body of response
   res.send(course);
})
```

Building web server – POST Requests

Handling HTTP POST Requests :- **app.post( )** • Postman can be used to make post requests	**// Post request to redirect to home route** app.post("/redirect", function(req, res){ res.redirect("/") })

Building web server – Input validation

Input validation with joi:-

- The joi module allows describing input validation logic using simple and readable language.
- https://joi.dev/api/

```
// load joi module
const Joi = require('joi');

app.post('/api/courses', (req, res) => {

  // define schema of object which defines properties of attributes
  const schema = Joi.object({
    // name is string & should have min. 3 characters and is required
    name: Joi.string().min(3).required()
  })

  // validate the sent object
  const Validation = schema.validate(req.body);

  // check is there is any error in object body
  if(Validation.error){
    res.status(400).send(Validation.error);
    return;
  }
}
```

Building web server – PUT Requests

Handling HTTP PUT Requests :-

app.put( )

- Postman can be used to make put requests

```
// Update a course
app.put("/api/courses/:id", (req, res) => {

  // look up the course & if not existing return 404
  const course = courses.find((c) => c.id === parseInt(req.params.id));
  if (!course) return res.status(404).send("Not found");

  // Validate, if invalid, return 400-bad request
  const {error} = validateCourse(req.body);   // equivalent  to result.error
  if (error)  return res.status(400).send(error.details[0].message);

  // Update course & return the updated course
  course.name = req.body.name;
  res.send(course); });

// Validate course function
function validateCourse(course) {
  const schema = Joi.object({
    name: Joi.string().min(3).required(), });

  return schema.validate(course);}
```

Building web server – DELETE Requests

Handling HTTP DELETE Requests :-

app.delete( )

- Postman can be used to make delete requests

```
// Delete a course
app.delete('/api/courses/:id', (req, res) => {

  // Look up the course, if not found then return 404
  const course = courses.find((c) => c.id === parseInt(req.params.id));
  if (!course) return res.status(404).send("Not found");

  // Delete course - find index & use splice method to remove
  const index = courses.indexOf(course);
  courses.splice(index, 1);

  // Return the deleted course
  res.send(course);
})
```

Working with form data

```
<form action="/" method="post">
    <input type="text" name="num1" placeholder="First Number">
    <input type="text" name="num2" placeholder="Second Number">
    <button type="submit" name="submit">Calculate</button>
</form>
```

Working with forms data :-

bodyParser.urlencoded()

- The following function parses the data coming from HTML form into a json object.
- The data entered in form can be thus accessed has if they are properties of objects

```
// Load the body_parser package
const bodyParser = require("body-parser");

// use() → get our app use a package
// bodyParser.urlencoded() --> Parse data coming from HTML form
// extended: true --> allows us to post nested objects
app.use(bodyParser.urlencoded({extended: true}))

// Tap into the form data
app.post("/", function(req, res){

   // req.body --> is parsed version of http request
   // { num1: '2', num2: '3', submit: '' }
   // we can tap into html form data as if they are properties of objects
   // Number → convert text into Numbers
   var num1 = Number(req.body.num1);
   var num2 = Number(req.body.num2);
   var result = num1 + num2;
   res.send(`The result of the calculation is ${result}`)
})
```

Rendering CSS and images

Rendering CSS & images :-

- Store your css & images at public folder to render them

// load our static files like css and images from public folder

app.use(express.static("public“));

href="css/signin.css“ (url to add for css)

src="images/Image.jpeg“ (url to add for image)

app.js template

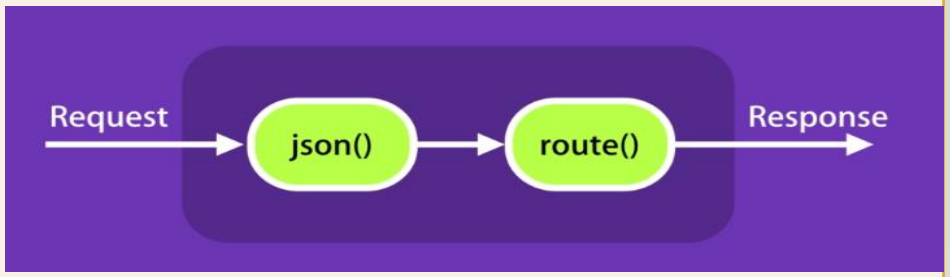
Here's a app.js template you can directly use every time→	const express = require("express"); const bodyParser = require("body-parser"); const app = express(); app.use(bodyParser.urlencoded({ extended: true })); app.use(express.static("public")); //app.set("view engine", "ejs"); app.get("/", function(req, res){ res.send("Hello"); }); app.listen(process.env.PORT || 3000, function(){ console.log("Server started on port 3000."); });

Middleware

Middleware Function :-

- Function that takes a request object & either terminates the request/response cycle or passes control to another middleware function

Routehandler in http requests & express.json() are examples of some middleware functions



Create custom middleware :-

- In the following example, next is a reference to next middleware function, if next() is not provided then our request will keep on hanging

```
// Create custom middleware function
app.use(function(req, res, next){
  console.log('Logging..')
  next();
})
```

Built-in middleware :-

- Express has a few built-in middleware functions

```
//enables parsing of json objects
express.json()

// parses incoming  requests with url encoded payloads
// extended: true, enables passing of arrays & complex objects
express.urlencoded({extended: true})

//Serves static files
express.static('folder_name')
```

Middleware

3rd party Middleware:-

- List of 3rd party middleware :-
- https://expressjs.com/en/resources/middleware.html

Examples of some 3rd party middleware functions :-

```
// Helmet middleware secures apps by setting various HTTP headers
const helmet = require('helmet');
app.use(helmet());

// Morgan is used to log HTTP requests
const morgan = require('morgan');
app.use(morgan('tiny'));
```

Environments

Environments :-	**// Returns environment for node app, undefined by default** process.env.NODE_ENV **// app object also returns environment variable, development by default** app.get('env') **// enable logging of http requests only on development environment** if (app.get('env') === 'development') { app.use(morgan('tiny')); console.log('morgan enabled...') } **// Set environment for node app on windows** set NODE_ENV=production **// Set environment for node app on MAC** export NODE_ENV=production

Configuration

Config package :-

- Set/store configuration settings & overwrite them in each environment
- Npm config package is used for manging configuration settings
- The custom-environment-variables can be used to store passwords related to database, etc..

// Create config folder in app directory & add default, development, production, custom-environment-variables etc.. Json file with their configuration settings

// Eg. config/development.json

```
{
   "name" : "my express app - dev",
    "mail" : {
      "host": "dev-mail-server"
    }
}
```

// The config module can be loaded in main app to set configuration settings for environment

```
const config =  require('config’);
```

// logs application name & mail host in terminal

```
console.log(`Application name: ${config.get('name')}`)
console.log(`Mail Server: ${config.get('mail.host')}`)
```

Debugging

Debug package :-

- The Console.log() function is frequently for debugging
- The debug package can be used for logging messages for debugging purposes

- Environment variable to determine what debugging messages to see →

```
// load debug module for arbitary namespace for debugging
const debuger = require('debug')('app:startup')
const debug = require('debug')('app:db’)

if (app.get('env') === 'development') {
  app.use(morgan('tiny’));
  debuger('morgan enabled...’)
}
debug("connected to database")

// see debugging messages for app:startup
set DEBUG=app:startup

 // see debugging messages for 2 or more namespace’s
set DEBUG=app:startup, app:db

// wildcard character to see debugging messages for namespaces starting with app:
set DEBUG=app:*

// clear debug env variable
Set DEBUG=
```

Templating Engines - PUG

Templating engine:-

- Templating engines are used to return html markup to client instead of json object
- Popular templating engines include pug, Mustache, EJS

```
// set view engine for our application to pug
app.set('view engine', 'pug');

// set path for our templates
app.set('views', './views')

// define html in pug
views/index.pug
html
  head
   title= title
  body
    h1= message

// render html template defined in index.js
// the 2nd argument refers to value of attributes
app.get("/", (req, res) => {
  res.render("index", {tittle: 'My Express App', message: 'Hello'});
});
```

Templating Engines - EJS

EJS :- • Embedded JavaScript templating • https://ejs.co/#install	**// Install EJS** npm i ejs
Using EJS :- • **<%= key %>** key refers value to be replaced with	**// set view engine for our application to ejs** app.set('view engine', 'ejs'); **// define html in ejs** **views/list.ejs** <title>To do list</title> It's a <%= kindOfDay %> **// render html defined in list.ejs** app.get("/", function (req, res) { var day = "weekday"; res.render("list", { kindOfDay: day }); });

Templating Engines - EJS

- **<% %>** :- For control flow

// Example of control flow in list.ejs
views/list.ejs

```
<% if (kindOfDay=="Saturday" || kindOfDay=="Sunday") { %>
<h1 style="color: purple"><%= kindOfDay %> ToDo List</h1>
<% } else { %>
<h1 style="color: blue"><%=kindOfDay%> ToDo List</h1>
<% } %>
```

EJS layouts :-

- **<%- include(‘ ’); -%>** syntax for including headers and footers
- header.ejs will contain the <head> part of html which contains all the link’s to stylesheets, meta description’s etc.
- footer.ejs will contain the <footer> part of html
- If header.ejs and footer.ejs is stored in partials folder then, syntax will be <%- include(“partials/header"); -%> etc.

header.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta/>
   <link/>
</head>
```

list.ejs

```
<%- include("header"); -%>
    <body>
   // main code goes here
   </body>
<%- include("footer"); -%>
```

footer.ejs

```
<footer>

 </footer>
</html>
```