Cheat Sheet: Developing Back-End Apps with Node.js and Express

Estimated reading time: 5 minutes

Package/Method	Description	Code Example

Async-await	We can await promises as long as they are being called inside asynchronous functions.	`const axios = require('axios').default;`
		`let url = "some remote url"`
		`async function asyncCall() {`
		`console.log('calling');`
		`const result = await axios.get(url);`
		`console.log(result.data);`
		`}`
		`asyncCall();`

callback	Callbacks are methods that are passed as parameters. They are invoked within the method to which they are passed as a parameter, conditionally or unconditionally.
	
	We use callbacks with a promise to process the response or errors.
		`//function(res) and function(err) are the anonymous callback functions`
		`axios.get(url)`
		`.then(function(res) {`
		`console.log(res);`
		`})`
		`.catch(function(err) {`
		`console.log(err)`
		`})`

Default/Date	new Date() method returns the current date as an object. You can invoke methods on the date object to format it or change the timezone.	`module.exports.getDate = function getDate() {`
		`var aestTime = new Date().toLocaleString("en-US", {timeZone: "Australia/Brisbane"});`
		`return aestTime;`
		`}`

express.get();	This method is meant to serve the retrieve requests to the server. The get() method is to be implemented with two parameters; the first parameter defines the end-point and the second parameter is a function taking the request-handler and response-handler.	`// handles GET queries to endpoint /user/about/id.`
		
		`app.get(“user/about/:id”, (req,res)=>{`
		`  res.send(“Response about user “`
		`  +req.params.id)`
		`})`

express.listen()	The listen method is invoked on the express object with the port number on which the server listens. The function is executed when the server starts listening.	`app.listen(3333, () => {`
		`console.log("Listening at http://localhost:3333)`
		`})`

express.post();	This method is meant to serve the create requests to the server. The post() method is to be implemented with two parameters: the first parameter defines the endpoint and the second parameter is a function taking the request-handler and response-handler.	`// handles POST queries to the same endpoint.`
		`app.post("user/about/:id", (req,res)=>{`
		`res.send("Response about user "`
		`+req.params.id)`
		`})`

express.Router()	Router-level middleware is not bound to the application. Instead, it is bound to an instance of express.Router(). You can use specific middleware for a specific route instead of having all requests going through the same middleware. Here, the route is /user, and you want the request to go through the user router. Define the router, define the middleware function that the router will use and what happens next, and then you bind the application route to the router.	`const express = require("express");`
		`const app = new express();`
		`var userRouter = expess.Router()`
		`var itemRouter = express.Router()`
		`userRouter.use(function (req, res, next){`
		`console.log("User quert time:", Date());`
		`next();`
		`})`
		`userRouter.get("/:id", function (req, res,`
		`next) {`
		`res.send("User "+req.params.id+ " last`
		`successful login "+Date())`
		`})`
		`app.listen(3333, () => {`
		`console.log("Listening at`
		`http://localhost:3333)`
		`})`

express.static()	This is an example of static middleware that is used to render static HTML pages and images from the server side. The static files can be rendered from the cad220_staticfiles directory at the application level. Notice that the URL has only the server address and the port number followed by the filename.	`const express = require("express");`
		`const app = new express();`
		`app.use(express.static("cad220_staticfiles`
		`"))`
		`app.listen(3333, () => {`
		`console.log("Listening at`
		`http://localhost:3333)`
		`})`

express.use()	This method takes middleware as a parameter. Middleware acts as a gatekeeper in the same order that it is used before the request reaches the get() and post() handlers. The order in which the middleware is chained depends on the order in which the .use() method is used to bind them. The middleware myLogger() function takes three parameters, which are request, response, and next. You can define a method that takes these three parameters and then bind it with express.use() or router.use(). Here, you are creating middleware named myLogger and making the application use it. The output rendered includes the time the request is received.	`const express = require("express");`
		`const app = new express();`
		`function myLogger(req, res, next){`
		`req.timeReceived = Date();`
		`next();`
		`}`
		`app.get("/", (req, res)=>{`
		`res.send("Request received at`
		`"+req.timeReceived+" is a success!")`
		`})`

express-react-views.createEngine() and express.engine()	This example uses express-react-views, which renders React components from the server. You set the view engine property, which is responsible for creating HTML from your views. Views are JSX code. The views are in a directory named myviews. The view engine will look for a JSX file named index in the myviews directory and pass the property name to it. The output rendered will have the name of the user.	`const express = require("express");`
		`const app = new express();`
		`const expressReactViews =`
		`require("express-react-views");`
		`const jsxEngine =`
		`expressReactViews.createEngine();`
		`app.set("view engine", "jsx");`
		`app.set("views", "myviews");`
		`app.engine("jsx", jsxEngine);`
		`app.get("/:name", (req, res)=>{`
		`res.render("index", { name:`
		`req.params.name });`
		`});`
		`app.listen(3333, () => {`
		`console.log("Listening at`

Package/Method	Description	Code Example
		http://localhost:3333) })

http/createServer	http package is used for establishing remote connections to a server or to create a server which listens to client. CreateServer - Takes a requestListener, a function which takes request and response parameters, where request is the handle to the request from the client and response is the handle to be sent to the client.	const http = require('http');
		const requestListener = function(req, res) {
		res.writeHead(200);
		res.end('Hello, World!');
		}
		const port = 8080;
		const server = http.createServer(requestListener);
		console.log('server listening on port: '+ port);

Import()	The import statement is used to import modules that some other module has exported. A file that includes reusable code is known as a module.	// addTwoNos.mjs
		function addTwo(num) {
		return num + 4;
		}
		
		export { addTwo };
		// app.js
		import { addTwoNos } from './addTwoNos.mjs';
		
		// Prints: 8
		console.log(addTwo(4));

new express()	Creates an express object which acts as a server application.	const express = require("express"); const app = new express();
Promise	An object that is returned by some methods, representing eventual completion or failure. The code continues to run without getting blocked until the promise is fulfilled or an exception is thrown.	axios.get(url) .then(//do something) .catch(//do something)

Promise use case	Promises are used when the processing time of the function we invoke takes time like remote URL access, I/O operations file reading, etc.	var prompt = require('prompt-sync')();
		var fs = require('fs');
		const methCall = new Promise((resolve,reject)=>{
		var filename = prompt('What is the name of the file ?');
		try {
		const data = fs.readFileSync(filename, {encoding:'utf8', flag:'r'});
		resolve(data);
		} catch(err) {
		reject(err)
		}
		});
		console.log(methCall);
		methCall.then(
		(data) => console.log(data),
		(err) => console.log("Error reading file")
		);

Require()	The built-in NodeJS method require() is used to incorporate external modules that are included in different files. The require() statement essentially reads and executes a JavaScript file before returning the export object.	module.exports = 'Hello Programmers'; message.js;var msg = require('./messages.js');console.log(message);

Skills Network