**30 Node.js Modules With Usecases –**

[Ravi Sharma](https://javascriptcentric.medium.com/?source=post_page-----36950203722f--------------------------------) Jul 4, 2022

A curated list of top useful Node.js modules that can help you in your upcoming backend projects.

NodeJS is an open-source platform for creating applications that use JavaScript on the server-side as well as on the client-side. Basically, It **allows us to run JavaScript code outside of the browser**. It is a runtime environment that interprets JavaScript using Google’s V8 JavaScript engine.

In this article, I am sharing a list of node modules that will help you in your node backend project. You can install these modules using npm.

Node.js’ package ecosystem (NPM), is the largest ecosystem of open source libraries in the world.

To install any node module run the below command in your terminal:

npm install module\_name

Here is a list of node modules that will extend the capability of your node.js application.

[**1. Express**](https://www.npmjs.com/package/express)**:** Express is a fast, un-opinionated, minimalist web framework. It provides small, robust tooling for HTTP servers, making it a great solution for single-page applications, websites, hybrids, or public HTTP APIs.

const express = require("express");const app = express();app.get("/", (req, res) => {  
 res.send("This is home page.");  
});app.post("/", (req, res) => {  
 res.send("This is home page with post request.");  
});const PORT = 3000;app.listen(PORT, () => {  
 console.log(`Server is running on PORT: ${PORT}`);  
});

[**2. Forever**](https://www.npmjs.com/package/forever)**:** A simple CLI tool for ensuring that a given node script runs continuously (i.e. forever).

forever start server.js

[**3. Nodemon**](https://www.npmjs.com/package/nodemon)**:** It is a simple monitor script for use during the development of a node.js app, It will watch the files in the directory in which nodemon was started, and if any files change, nodemon will automatically restart your node application.

nodemon server.js

[**4. Helmet:**](https://www.npmjs.com/package/helmet)Helmet middleware is a toolkit that helps you to secure your Express apps by setting various HTTP headers.

const express = require('express');const helmet = require("helmet");const app = express();app.use(helmet());// or You can use individual headersapp.use(helmet.contentSecurityPolicy());app.use(helmet.crossOriginEmbedderPolicy());app.use(helmet.crossOriginOpenerPolicy());app.use(helmet.crossOriginResourcePolicy());app.use(helmet.dnsPrefetchControl());app.use(helmet.expectCt());app.use(helmet.frameguard()); // Prevent Click Jacking Attackapp.use(helmet.hidePoweredBy()); // Disable tech-stack from headerapp.use(helmet.hsts()); // Set strict transport securityapp.use(helmet.ieNoOpen());app.use(helmet.noSniff());app.use(helmet.originAgentCluster());app.use(helmet.permittedCrossDomainPolicies());app.use(helmet.referrerPolicy());app.use(helmet.xssFilter());// Prevent Cross-site scripting attack

[**5. Cors:**](https://www.npmjs.com/package/cors)CORS is shorthand for Cross-Origin Resource Sharing. It is a mechanism to allow or restrict requested resources on a web server depends on where the HTTP request was initiated.  
This policy is used to secure a certain web server from access by other websites or domains. For example, only the allowed domains will be able to access hosted files in a server such as a stylesheet, image, or script.

const express = require('express')  
const cors = require('cors')  
const app = express()//Simple Usage (Enable All CORS Requests)  
app.use(cors())//Enable CORS for a Single Route  
app.get('/products/:id', cors(), function (req, res, next) {  
 res.json({msg: 'This is CORS-enabled for a Single Route'})  
})

[**6. Moment**](https://www.npmjs.com/package/moment)**:** A lightweight JavaScript date library for parsing, validating, manipulating, and formatting dates.

var now = "04/09/2013 15:00:00";  
var then = "02/09/2013 14:20:30";var ms = moment(now, "DD/MM/YYYY HH:mm:ss").diff(moment(then, "DD/MM/YYYY HH:mm:ss"));var d = moment.duration(ms);  
var s = d.format("hh:mm:ss");

[**7. Morgan:**](https://www.npmjs.com/package/morgan)HTTP request logger middleware for node.js. Including the preset tiny as an argument to morgan() will use its built-in method, identify the URL, declare a status, and the request’s response time in milliseconds.

const express = require('express');const morgan = require('morgan');const app = express();app.use(morgan('tiny'));

[**8. Validator**](https://www.npmjs.com/package/validator)**:** A node module for a library of string validators and sanitizers.

const validator = require('validator')// Check whether given email is valid or not  
var email = '[test@gmail.com](mailto:test@gmail.com)'  
console.log(validator.isEmail(email)) // trueemail = 'test@'  
console.log(validator.isEmail(email)) // false// Check whether string is in lowercase or not  
var name = 'geeksforgeeks'  
console.log(validator.isLowercase(name)) // truename = 'GEEKSFORGEEKS'  
console.log(validator.isLowercase(name)) // false// Check whether string is empty or not  
var name = ''  
console.log(validator.isEmpty(name)) // truename = 'geeksforgeeks'  
console.log(validator.isEmpty(name)) // false// Other functions also available in   
// this module like isBoolean()  
// isCurrency(), isDecimal(), isJSON(),  
// isJWT(), isFloat(), isCreditCard(), etc.

[**9. Async**](https://www.npmjs.com/package/async)**:** Async is a utility module that provides straight-forward, powerful functions for working with [asynchronous JavaScript](http://caolan.github.io/async/v3/global.html).

Many helper methods exist in Async that can be used in different situations, like **series, parallel, waterfall**, etc. Each function has a specific use case, so take some time to learn which one will help in which situations.

async.series([  
 function(callback) {  
 // do some stuff ...  
 callback(null, 'one');  
 },  
 function(callback) {  
 // do some more stuff ...  
 callback(null, 'two');  
 }  
 ],  
 // optional callback  
 function(err, results) {  
 // results is now equal to ['one', 'two']  
 }  
);//============================================================async.parallel({  
 one: function(callback) {  
 ...  
 },  
 two: function(callback) {  
 ...  
 },  
 ...  
 something\_else: function(callback) {  
 ...  
 }  
 },  
 // optional callback  
 function(err, results) {  
 // 'results' is now equal to: {one: 1, two: 2, ..., something\_else: some\_value}  
 }  
);//============================================================async.waterfall([  
 function(callback) {  
 callback(null, 'one', 'two');  
 },  
 function(arg1, arg2, callback) {  
 // arg1 now equals 'one' and arg2 now equals 'two'  
 callback(null, 'three');  
 },  
 function(arg1, callback) {  
 // arg1 now equals 'three'  
 callback(null, 'done');  
 }  
], function(err, result) {  
 // result now equals 'done'  
});

[**10. Mongoose**](https://www.npmjs.com/package/mongoose)**:** It is a MongoDB ODM (object data modeling) tool designed to work in an asynchronous environment. this package enables you to easily connect to a MongoDB database using Node.js.

const mongoose = require('mongoose');const connectDB = async () => {  
 mongoose  
 .connect('mongodb://localhost:27017/playground', {  
 useCreateIndex: true,  
 useNewUrlParser: true,  
 useUnifiedTopology: true,  
 useFindAndModify: false  
 })  
 .then(() => console.log('Connected Successfully'))  
 .catch((err) => console.error('Not Connected'));  
} module.exports = connectDB;

[**11. Mysql:**](https://www.npmjs.com/package/mysql)The *mysql* package enables you to easily connect to a MySQL database using Node.js.

var mysql = require('mysql');var connection = mysql.createConnection({  
 host : 'localhost',  
 database : '*dbname*',  
 user : '*username*',  
 password : '*password*',  
});  
  
connection.connect(function(err) {  
 if (err) {  
 console.error('Error connecting: ' + err.stack);  
 return;  
 }  
  
 console.log('Connected as id ' + connection.threadId);  
});  
  
connection.query('SELECT \* FROM employee', function (error, results, fields) {  
 if (error)  
 throw error;  
  
 results.forEach(result => {  
 console.log(result);  
 });  
});  
  
connection.end();

[**12. Nodemailer**](https://www.npmjs.com/package/nodemailer)**:** This module enables e-mail sending from Node.js applications.

"use strict";  
const nodemailer = require("nodemailer");async function main() {  
 let testAccount = await nodemailer.createTestAccount();  
 let transporter = nodemailer.createTransport({  
 host: "smtp.ethereal.email",  
 port: 587,  
 secure: false, // true for 465, false for other ports  
 auth: {  
 user: testAccount.user, // generated ethereal user  
 pass: testAccount.pass, // generated ethereal password  
 },  
 }); let info = await transporter.sendMail({  
 from: '"Fred Foo 👻" <[foo@example.com](mailto:foo@example.com)>', // sender address  
 to: "[bar@example.com](mailto:bar@example.com), [baz@example.com](mailto:baz@example.com)", // list of receivers  
 subject: "Hello ✔", // Subject line  
 text: "Hello world?", // plain text body  
 html: "<b>Hello world?</b>", // html body  
 });console.log("Message sent: %s", info.messageId);  
 // Message sent: <[b658f8ca-6296-ccf4-8306-87d57a0b4321@example.com](mailto:b658f8ca-6296-ccf4-8306-87d57a0b4321@example.com)> console.log("Preview URL: %s", nodemailer.getTestMessageUrl(info));  
   
}main().catch(console.error);

[**13. Bcrypt:**](https://www.npmjs.com/package/bcrypt)The bcrypt NPM package is **a JavaScript implementation of the bcrypt password hashing function that allows you to easily create a hash out of a password string**.

Hashing is a one-way ticket to data encryption. Hashing performs a one-way transformation on a password, turning the password into another String, called the hashed password. Hashing is called one way because it’s practically impossible to get the original text from a hash.

const bcrypt = require("bcrypt");const express = require("express");const User = require("./userModel");const router = express.Router();// signup route  
router.post("/signup", async (req, res) => {  
 const body = req.body;if (!(body.email && body.password)) {  
 return res.status(400).send({  
 error: "Data not formatted properly"  
 });  
 }// creating a new mongoose doc from user data  
 const user = new User(body); // generate salt to hash password  
 const salt = **await bcrypt.genSalt(10);** // now we set user password to hashed password  
 user.password = **await bcrypt.hash(user.password, salt);**  
 user.save().then((doc) => res.status(201).send(doc));  
});// login route  
router.post("/login", async (req, res) => {  
 const body = req.body;  
 const user = await User.findOne({  
 email: body.email  
 });  
 if (user) {  
 // check user password with hashed password stored in the database  
 const validPassword = **await bcrypt.compare(body.password, user.password);**  
 if (validPassword) {  
 res.status(200).json({  
 message: "Valid password"  
 });  
 } else {  
 res.status(400).json({  
 error: "Invalid Password"  
 });  
 }  
 } else {  
 res.status(401).json({  
 error: "User does not exist"  
 });  
 }  
});module.exports = router;module.exports = router;

[**14. Express-rate-limit:**](https://www.npmjs.com/package/express-rate-limit)Basic rate-limiting middleware for Express. Use to limit repeated requests to public APIs and/or endpoints such as password reset.

const express = require('express');const rateLimit = require('express-rate-limit');const limiter = rateLimit({  
 windowMs: 15 \* 60 \* 1000, // 15 minutes  
 max: 100, // Limit each IP to 100 requests per `window` (here, per 15 minutes)  
 standardHeaders: true, // Return rate limit info in the `RateLimit-\*` headers  
 legacyHeaders: false, // Disable the `X-RateLimit-\*` headers  
})// Apply the rate limiting middleware to all requests  
app.use(limiter)

[**15. Response-time:**](https://www.npmjs.com/package/response-time)This module creates a middleware that records the response time for requests in HTTP servers. The “response time” is defined here as the elapsed time from when a request enters this middleware to when the headers are written out to the client.

const express = require('express');const responseTime = require('response-time');const app = express()app.use(responseTime())app.get('/', function(req, res) {  
 res.send('hello, world!')  
})

[**16. connect-busboy:**](https://www.npmjs.com/package/connect-busboy)busboyis astreaming parser for HTML form data for node.js.

const express = require('express');const busboy = require('connect-busboy');const path = require('path');const fs = require('fs-extra');const app = express();app.use(**busboy({  
 highWaterMark: 2 \* 1024 \* 1024, // Set 2MiB buffer  
})**);const uploadPath = path.join(\_\_dirname,  
 fs.ensureDir(uploadPath);app.route('/upload').post((req, res, next) => {**req.pipe(req.busboy);** // Pipe it trough busboy**req.busboy.on**('file', (fieldname, file, filename) => {  
 console.log(`Upload of '${filename}' started`);const fstream = fs.createWriteStream(path.join(uploadPath, filename));// Pipe it trough  
 file.pipe(fstream);  
 fstream.on('close', () => {  
 console.log(`Upload of '${filename}' finished`);  
 res.redirect('back');  
 });  
 });  
 });const server = app.listen(3200, function() {  
 console.log(`Listening on port ${server.address().port}`);  
 });

[**17. Google-auth-library:**](https://www.npmjs.com/package/google-auth-library)This is Google’s officially supported node. js client library for using OAuth 2.0 authorization and authentication with Google APIs.

const { OAuth2Client } = require("google-auth-library");async function googleSignInUser(request, response) {  
 const client = new OAuth2Client(process.env.GOOGLE\_CLIENT\_ID);  
 const { idToken } = request.body;client  
 .verifyIdToken({ idToken, audience: process.env.GOOGLE\_CLIENT\_ID })  
 .then((res) => {  
 const { email\_verified, name, email } = res.payload;  
 if (email\_verified) {  
 User.findOne({ email }).exec((err, user) => {  
 if (user) {  
 const { \_id, email, fullName } = user;const token = jwt.sign({ email: email }, process.env.SECRET\_KEY, {  
 expiresIn: process.env.EXPIRE\_IN,  
 });return response.status(200).json({  
 accessToken: token,  
 user: { \_id, email, fullName },  
 });  
 } else {  
 const password = email + process.env.SECRET\_KEY;bcrypt.hash(password, 12, async (err, passwordHash) => {  
 if (err) {  
 response.status(500).send("Couldn't hash the password");  
 } else if (passwordHash) {  
 return User.create({  
 email: email,  
 fullName: name,  
 hash: passwordHash,  
 }).then((data) => {  
 const { \_id, email, fullName } = data;  
 const token = jwt.sign(  
 { email: email },  
 process.env.SECRET\_KEY,  
 { expiresIn: process.env.EXPIRE\_IN }  
 );response.status(200).json({  
 accessToken: token,  
 user: { \_id, email, fullName },  
 });  
 });  
 }  
 });  
 }  
 });  
 } else {  
 return res.status(400).json({  
 error: "Google login failed. Try again",  
 });  
 }  
 });  
}

[**18. Redis:**](https://www.npmjs.com/package/redis)Redis is a super fast and efficient in-memory, key-value cache and store. It’s also known as a data structure server, as the keys can contain strings, lists, sets, hashes, and other data structures.

const redis = require("redis");const client = createClient();client.on('error', (err) => console.log('Redis Client Error', err));await client.connect();await client.set('key', 'value');  
const value = await client.get('key');

[**19. Joi:**](https://www.npmjs.com/package/joi)The most powerful schema description language and data validator for JavaScript.

const Joi = require('joi'); app.post('/blog', async (req, res, next) => {   
 const { body } = req; const blogSchema = **Joi.object().keys**({   
 title: **Joi.string().required**   
 description: **Joi.string().required(),**   
 authorId: **Joi.number().required()**   
 }); const result = Joi.validate(body, blogShema);   
 const { value, error } = result;   
 const valid = error == null; if (!valid) {   
 res.status(422).json({   
 message: 'Invalid request',   
 data: body   
 })   
 } else {   
 const createdPost = await api.createPost(data);   
 res.json({ message: 'Resource created', data: createdPost })   
 }   
});

[**20. Winston:**](https://www.npmjs.com/package/winston)Winston, is one of the best logging middleware. Logging is a process of recording information generated by application activities into log files. Messages saved in the log file are called logs. A log is a single instance recorded in the log file.

A log is the first place to look as a programmer, to track down errors and flow of events, especially from a server. A log tells you what happens when an app is running and interacting with your users. A great use case for logging would be if, for example, you have a bug in your system, and you want to understand the steps that led up to its occurrence. Let's take an example of the custom logger.js

const { createLogger, format, transports, config } = require('winston');  
  
const usersLogger = createLogger({  
  
 levels: config.syslog.levels,  
 format: combine(  
 timestamp({  
 format: 'YYYY-MM-DD HH:mm:ss'  
}),  
  
 transports: [  
 new transports.File({ filename: 'users.log' })  
 ]  
});  
const transactionLogger = createLogger({  
 transports: [  
 new transports.File({ filename: 'transaction.log' })  
 ]  
});  
  
module.exports = {  
 usersLogger: usersLogger,  
 transactionLogger: transactionLogger  
};

[**21. node-fetch:**](https://www.npmjs.com/package/node-fetch)A light-weight module that brings Fetch API to node.js.

const fetch = require('node-fetch');fetch('<https://api.github.com/users/github'>)  
 .then(res => res.json())  
 .then(json => console.log(json));

[**22. WS (WebSocket library):**](https://www.npmjs.com/package/ws) It is a simple to use, blazing-fast, and thoroughly tested WebSocket client and server implementation.

const WebSocketServer = require('ws');const wss = new WebSocketServer({  
 port: 8080  
});wss.on('connection', function connection(ws) {  
 ws.on('message', function message(data) {  
 console.log('received: %s', data);  
 });ws.send('something');  
});

[**23. loadtest:**](https://www.npmjs.com/package/loadtest)Runs a load test on the selected HTTP or WebSockets URL. The API allows for easy integration in your own tests.

$ loadtest [-n requests] [-c concurrency] [-k] URL  
$ loadtest -n 100000 -c 10000 <http://localhost:9090/>

[**24. i18next:**](https://www.npmjs.com/package/i18next)i18next is a very popular internationalization framework for browsers or any other JavaScript environment (eg. Node.js, Deno).

const http = require('http');const path = require('path');const { I18n } = require('i18n');  
  
const i18n = new I18n({  
 locales: ['en', 'de'],  
 directory: path.join(\_\_dirname, 'locales')  
})  
  
const app = http.createServer((req, res) => {  
 i18n.init(req, res)  
 res.end(res.\_\_('Hello'))  
})  
  
app.listen(3000, '127.0.0.1')

[**25. jsonwebtoken:**](https://www.npmjs.com/package/jsonwebtoken) JWT, or JSON Web Token, is an open standard used to share security information between two parties — a client and a server.

app.post("/login", async (req, res) => {  
 try {  
 const {  
 email,  
 password  
 } = req.body;// Validate user input  
 if (!(email && password)) {  
 res.status(400).send("All input is required");  
 }  
 // Validate if user exist in our database  
 const user = await User.findOne({  
 email  
 });if (user && (await bcrypt.compare(password, user.password))) {  
   
// Create token  
 const token = **jwt.sign**({  
 user\_id: user.\_id,  
 email  
 },  
 process.env.TOKEN\_KEY, {  
 expiresIn: "2h",  
 }  
 );// save user token  
 user.token = token;// user  
 res.status(200).json(user);  
 }  
 res.status(400).send("Invalid Credentials");  
 } catch (err) {  
 console.log(err);  
 }  
 // Our register logic ends here  
});

[**26. Cookie-parser:**](https://www.npmjs.com/package/cookie-parser)cookie-parser is a middleware that parses cookies attached to the client request object. To use it, we will require it in our index. js file; this can be used the same way as we use other middleware

const Express = require('express');  
const app = Express();  
const port = 80;**const CookieParser = require('cookie-parser');  
app.use(CookieParser());**app.get("/send", (req, res) => {  
 res.cookie("loggedin", "true");  
 res.send("Cookie sent!");  
});app.get("/read", (req, res) => {let response = "Not logged in!";if (req.cookies.loggedin == "true") {  
 response = "Yup! You are logged in!";  
 }res.send(response);  
});app.listen(port, () => {  
 console.log("Server running!");  
});

[**27. Config:**](https://www.npmjs.com/package/config) Node-config organizes hierarchical configurations for your app deployments.

npm install config

Create a config directory and add a config/default.json file to it. This will be the default config file and will contain all your default environment variables.

{  
 "server": {  
 "host": "localhost",  
 "port": 8080,  
 }  
 }

To use the config file :

const express = require('express');  
const config = require('config');  
const app = express();  
const port = config.get('server.port');  
const host = config.get('server.host');  
  
app.get('/', (req, res) => {  
 res.send('Hello World');  
});  
const server = app.listen(port, host, (err) => {  
 if (err) {  
 console.log(err);  
 process.exit(1);  
 }  
 console.log(`Server is running on ${host}:${server.address().port}`);  
});

[**28. Supertest:**](https://www.npmjs.com/package/supertest)SuperTest is **a Node.** **js library that helps developers test APIs**. It extends another library called superagent, a JavaScript HTTP client for Node. js and the browser. Developers can use SuperTest as a standalone library or with JavaScript testing frameworks like Mocha or Jest.

const request = require('supertest');  
const app = require('/app');describe('Testing POSTS/shots endpoint', function() {  
 it('respond with valid HTTP status code and description and message', function(done) {  
 const response = await request(app).post('/shots').send({  
 title: 'How to write a shot',  
 body: "Access the Edpresso tutorial"  
 });expect(response.status).toBe(200);  
 expect(response.body.status).toBe('success');  
 expect(response.body.message).toBe('Shot Saved Successfully.');  
 });  
});

[**29.Multer:**](https://www.npmjs.com/package/multer)Multer is a node.js middleware for handling, which is primarily used for uploading files. It is written on top of the [busboy](https://github.com/mscdex/busboy) for maximum efficiency.

// upload.js  
const multer = require("multer");  
const path = require("path");const storage = multer.diskStorage({  
 destination: (req, file, cb) => {  
 cb(null, "./public/uploads/images/");  
 },  
 filename: (req, file, cb) => {  
 cb(null, Date.now() + file.originalname);  
 },  
});const fileFilter = (req, file, cb) => {  
 if (file.mimetype === "image/jpeg" || file.mimetype === 'image/jpg' || file.mimetype === "image/png") {  
 cb(null, true);  
 } else {  
 cb(null, false);  
 }  
};module.exports = multer({  
 storage: storage,  
 limits: {  
 fileSize: 1024 \* 1024 \* 5,  
 },  
 fileFilter: fileFilter,  
});// Use this middleware   
app.post('/uploadfile', upload.single('myFile'), (req, res, next) => {  
 const file = req.file  
 if (!file) {  
 const error = new Error('Please upload a file')  
 error.httpStatusCode = 400  
 return next(error)  
 }  
 res.send(file)})

[**30. Compression:**](https://www.npmjs.com/package/compression) It is a Node.js compression middleware. Compression in Node. js and Express decreases the downloadable amount of data that is served to users.

const compression = require('compression');const express = require('express');const app = express()// compress all responses  
app.use(compression())