Express – Advance Topics

1. Middleware
2. Configurations
3. Debugging
4. Template Engine

# Middleware

Middleware or Middleware function is basically a function that takes a request object , and either return a response to the client or passes control to another Middleware function.

We already have seen two middleware function

1. This route handler function

app.get('/', (req, res) => {

res.send('Welcome to Vidly');

});

In express every route function is technically a Middleware function, because it takes a request object and return a response to a client. So it terminate the request response cycle.

1. Another middleware function

app.use(express.json());

So when we call express.json method this method returns a function, a Middleware function. The job of this middleware function is to read the request and if there is a json object in the body of the request .It will parse the body of the request into json object and then it set the request.body property. So essentially this happens at the runtime.

A picture containing drawing

Description automatically generated

# Create a Custom Middleware Function

//custom middleware function logging

app.use(function (req, res, next) {

console.log('Logging...');

next();

});

//custom middleware function authenticating

app.use(function (req, res, next) {

console.log('Authenticating...');

next();

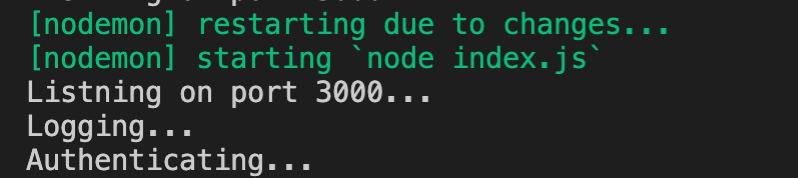
});

The Middleware function are executed in a sequence they are defined.

GET http://localhost:3000/api/courses/

A screenshot of a cell phone

Description automatically generated



Moving the Middleware code to a separate file

logger.js

function log(req, res, next) {

console.log('Logging...');

next();

}

module.exports = log;

authenticating.js

function authenticate(req, res, next) {

console.log('Authenticating...');

next();

}

module.exports = authenticate;

index.js

const logger = require('./logger');

const authenticate = require('./authenticating');

//adding a piece of middleware

app.use(express.json());

//custom middleware function logging

app.use(logger);

//custom middleware function authenticating

app.use(authenticate);

GET http://localhost:3000/api/courses/

A screenshot of a cell phone

Description automatically generated

A picture containing green, white

Description automatically generated

# Built-In Middleware

//built in middleware function

app.use(express.urlencoded({ extended: true }));

POST http://localhost:3000/api/courses/

Content-Type: application/x-www-form-urlencoded

name=my-course

A screenshot of a cell phone

Description automatically generated

A picture containing sitting, room

Description automatically generated

//built in middleware function

app.use(express.static('public'));

create a public folder

Inside public folder .Create a file readme.txt

|  |
| --- |
| This is a readme file! |

GET http://localhost:3000/readme.txt

A screenshot of a cell phone

Description automatically generated

# Third-party Middleware

Go to : <http://expressjs.com/>

In the resources we have middleware

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

We will look for helmet

A screenshot of a cell phone

Description automatically generated

$ npm i helmet

$ npm i morgan

Index.js

//Third-party middleware

const helmet = require('helmet')

const morgan = require('morgan')

//third-party middleware function

app.use(helmet());

app.use(morgan('tiny'));

GET http://localhost:3000/api/courses/

A close up of a sign

Description automatically generated

This is morgan(‘tiny’) format: GET /api/courses/ 200 79 - 3.226 ms

# Environments

Is this a development environment or production environment. You want to enable or disable certain features based on the current environment.

Let’s say you want to enable logging up http request only in development environment

app.use(morgan('tiny'));

//Enviornments

console.log(`NODE\_ENV: ${process.env.NODE\_ENV}`);

console.log(`app: ${app.get('env')}`);

if (app.get('env') === 'development') {

app.use(morgan('tiny'));

console.log('Morgan enabled...');

};

By default : its development

A screenshot of a cell phone

Description automatically generated

You can see the NODE\_ENV is undefine. Let set the NODE\_ENV =production

A screenshot of a cell phone

Description automatically generated

Current env is production. You can see the if condition which check for development environment. The Morgan enabled is skipped.

# Configuration

In the last topic we learnt which environment you application is running. One topic that go hand in hand is the topic of storing configuration settings for the application.And overriding those settings in each environment.

For example, in your development environment you’re going to use a different database for mail server. In this module we’ll discuss how to do configuration setting for your applications and override them in each environment.

There are various node packages out there for managing configurations.Most populat is rc.

<https://www.npmjs.com/package/rc>

npm config

<https://www.npmjs.com/package/config>

# Debugging

Console.log is a javascript programmer friend. We use this for debugging. But the problem is that sometimes when we are done with them . we comment our or delete them. But latter we may again need it we have to remove the comments or write them again. It’s a tedious process. But way to log message for the purpose of debugging is to use the debugging package in node. With debug we are going to replace all our console.log statements that are called to debug function and then we can use environment variables to enable or disable debugging. This way we don’t have to come back to our code and modify the code. We don’t have to delete this console.log statements. We don’t have to comment them out. We can control them from outside using an environment. But more importantly we can also determine the level of debugging information we want to see. May be sometime you are working on the database problem , or house . we only want to see a debugging information related to database problem. Again we don’t have to come back to our code and modify all these console.log statements.

Install debug pagkage

$ npm i debug

const startupDebugger = require('debug')('app:startup');

const dbDebugger = require('debug')('app:database');

//third-party middleware function

app.use(helmet());

if (app.get('env') === 'development') {

app.use(morgan('tiny'));

//console.log('Morgan enabled...');

startupDebugger('Morgan enabled...');

};

//Db work...

dbDebugger('Connected to the database');

No go to console to set the Debugger information

$export DEBUG=app:db

$nodemon index.js

A screenshot of a cell phone

Description automatically generated

$export DEBUG=

$nodemon index.js

A screenshot of a cell phone

Description automatically generated

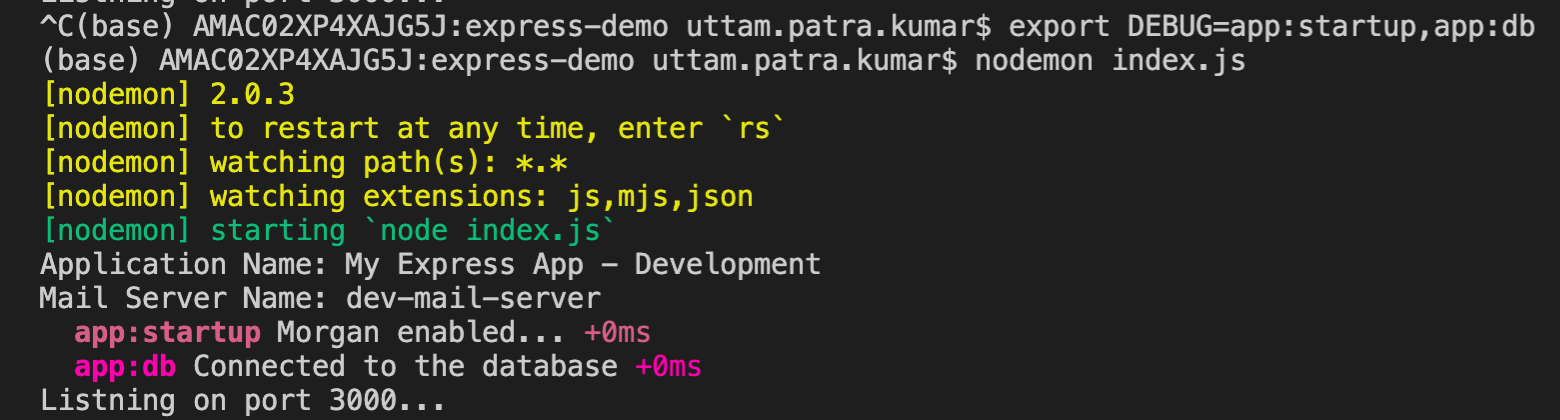
$export DEBUG=app:startup

$nodemon index.js

A screenshot of a cell phone

Description automatically generated

$export DEBUG=app:startup,app:db



$export DEBUG=app:\*

A screenshot of a cell phone

Description automatically generated

$ DEBUG=app:db nodemon index.js

A screenshot of a cell phone

Description automatically generated

In a real time, application, we may not have multiple debugger

const debug = require('debug')('app:startup');

//const dbDebugger = require('debug')('app:db');

//Third-party middleware

const helmet = require('helmet');

const morgan = require('morgan');

//third-party middleware function

app.use(helmet());

if (app.get('env') === 'development') {

app.use(morgan('tiny'));

//console.log('Morgan enabled...');

debug('Morgan enabled...');

};

A screenshot of a cell phone

Description automatically generated

# TEMPLATE ENGINES

So far we have returned json object in response. Some time you may need to send html mark-up to the client.Thats why use template engine. There are various template engine available for express applications.Most popular are PUG, Mustache, EJS. Each templating engine has different syntax for generating dynamic html and returning to client. In this demo we will see how to use PUG to generate a dynamic html and returning it to the client.

Install PUG

npm install pug

create a view folder

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Inside view folder create a file index.pug

html

head

title= title

body

h1= message

inside index.js

app.set('view engine', 'pug');

app.set('views', './views');//default

app.get('/', (req, res) => {

//callback function called route handler

//res.send('Hello World !!!');

res.render('index', { title: 'My Express Page', message: 'Welcome to the website !!' });

})

$ nodemon index.js

