**EXPRESS JS**

# 1) Cài đặt Express JS trong Project

npm init

npm install express --save

# 2) Khởi tạo Express

const express = require('express')

const app = express()

const port = 3000

app.listen(port, function() {

console.log(`Example app listening on port + port)

})

# 3) Route

* app.get('/path', function(req, res){

res.send(‘<h1>Hello world</h1>’) ;

})

*Trong đó :*

/path: Đường dẫn chỉ định

req: Chứa tất cả thông số mà khách hàng truyền lên

res: Chứa tất cả thông số mà server tải về

send: Method của res

=> Method **get**: Khi có 1 get request gửi lên /path thì function callback với 2 tham số req,res sẽ được gọi và method res.send sẽ trả về 1 cái gì đó

# 4) Template engine

## a) Setup:

* npm install pug --save
* ***Cài đặt template engine:*** app.set('view engine', 'pug')

pug: Template engine sử dụng

* ***Cài đặt thư mục gốc:*** app.set('views', './views')

./views: Thư mục gốc chỉ định

## b) Use

* app.get('/path', function (req, res) {

res.render('index', { title: 'Hey', message: 'Hello there!' })

})

*Trong đó :*

/path: Đường dẫn chỉ định

res.render: Method này sẽ compile pug sang html

index: Tên của file pug (Tham số thứ nhất)

{title: 'Hey', message: 'Hello there!'}: Là một object truyền. Chúng ta có thể lấy trực tiếp giá trị từ biến này ở trong index (Tham số thứ hai không bắt buộc)

NOTE: method render nhận vào 2 tham số

# 5) Query parameters



Đoạn mã trên nhằm mục đích tìm kiếm trong danh sách những thứ liên quan đến “my tam”

## a) Use

* req.query.q: để get giá trị của q
* form(action="/users/search" method="GET")

input (type="text" name="q")

button

Đoạn code trên nhằm mục đích gửi lên đường dẫn giá trị của ***q*** người dùng nhập ở trong input

# 6) Post method

## a) Use

* req.body: Trả về 1 cặp key value trong dữ liệu được gửi đi. Mặc định là undefined, ta có thể dùng body-parsing middleware như **body-parse** để phân tích nó ra các cặp key value nằm trong object

Cài đặt **body-parse:**

- npm install body-parse --save

- app.use(bodyParser.json());

- app.use(bodyParser.urlencoded({ extended: true }))

* Ví dụ:

app.post('/path', function (req, res) {

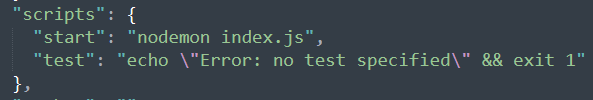
console.log(req.body)

})

# 7) Nodemon

## a) Install

* npm install --save-dev nodemon
* Thêm vào file package.jsom như sau:



## b) Use

Như vậy khi start server ta chỉ cần chạy lệnh nodemon index.js thì terminal sẽ tự động reload lại khi có thay đổi trên file index

# 8) Lowdb

## a) Install

* npm install lowdb
* const low = require('lowdb')
* const FileSync = require('lowdb/adapters/FileSync')
* const adapter = new FileSync('db.json')
* const db = low(adapter)

## b) Use

* db.defaults({ db: []})

.write()

db: [] là giá trị của 1 array mặc định mà sau đó ta sẽ truyền dữ liệu trong file db.json vào

NOTE: Nếu project có dùng nodemon thì phải tạo 1 file nodemon.json và ignore file db.json

* db.get('db').value(): Trả về giá trị trong file db.json

value(): Method để lấy giá trị bên trong file json

db: Tên của file database.json

db: Tên của giá trị mặc định ta đặt lúc đầu

* Ghi dữ liệu vào file database:

db.get('db ').push({name: John}).write()

# 9) View user

## a) Use

* Route path: /users/:userId

Request URL: http://localhost:3000/users/34/

req.params: { "userId": "34"}

* router.get('path/:id', function(req,res){

var id = req.params.id;

var user = db.get('users').find({id: id}).value()

res.render('users/view', {

user: user

})

})

*Trong đó:*

path/:id: Cứ khi nào có 1 get request gửi đến /path/một id nào đó thì chạy callback

req.params.id: Một id nào đó được lưu vào biến này

db.get('users').find({id: id}).value(): Method của lowdb sẽ tìm kiếm và lấy giá trị id trùng với id được

## b) Create id using shortId

* Install: npm install shortid --save
* Use:

Khai báo thư viện: var ids **=** require('short-id');

Sử dụng: ids.generate()

# 10) Express Router

- Dùng để chia nhỏ chương trình thành những folder riêng biệt chứa các file có chức năng giống nhau

## a) Use

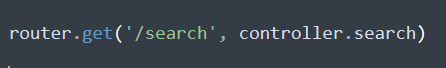
* Tạo 1 folder router bên trong chứa file user.router.js
* Khai báo router: var router = express.Router()
* Để những thứ liên quan vào trong 1 file js (VD: app.get(/user/…) thì cho những gì liên quan đến user vào 1 file user.route.js
* Thay thế app. = router. , đồng thời xóa /user
* Sau đó export file js ra bằng lệnh module.exports = user;
* Trong file main.js require file moudle user bằng lệnh var userRoutes = require('./routes/user.route')
* Sau đó sửa callback đằng sau của app thành app.use('/users', userRoutes);

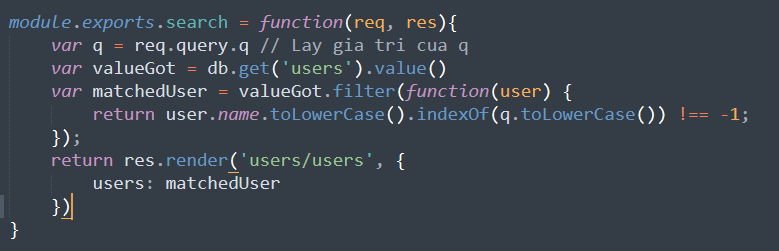
NOTE: Khi chia nhỏ file ra mà file có có yêu cầu module cần thiết thì phải require module đó vào

# 11) Controller

- Những thứ liên quan đến logic xử lý sẽ được tách riêng là 1 file controller

## a) Use

* Tạo 1 folder controllers bên trong chứa file user.controller.js
* Trong file user.router.js require module controller vào
* Trong file user.router.js sửa callback thành
* Trong file user.controller.js viết



# 12) Static file

## a) Use

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

public: Thư mục public này chứa các file tĩnh như css, js, images …

* Sau khi đã khai báo ta chỉ cần dẫn link /css /js /images là được

# 13) Middleware

## a) Define

* Middleware là 1 function nhận vào 3 tham số là **req**,**res**,**next** nhưng thông thường chúng ta k dùng đến next nên chúng ta sẽ bỏ đi
* Có thể dùng nhiều middleware trong 1 route
* Trong route các middleware sẽ được chạy theo thứ tự
* Nếu có hàm next() trong middleware thì các middleware được gọi phía sau sẽ được thực hiện

## b) Use

* Eg:

function middleware1(req,res,next){

next();

}

function middleware2(req,res,next){

next();

}

route.get('/test', middleware1, middleware2)

# 14) Cookie

## a) Define

* **Cookie là một đoạn văn bản mà một Web server có thể lưu trên ổ cứng của người dùng. Cookie cho phép một website lưu các thông tin trên máy tính của người dùng và sau đó lấy lại nó. Các mẩu thông tin sẽ được lưu dưới dạng cặp tên – giá trị (name-value).**

## b) Use

* res.cookie(name, value [, options])
* Eg: router.get('/cookie', function(req,res,next){

res.cookie('user-id', 12345)

})

# 15) Signed cookie

## a) Define

* Dùng để bảo mật cookie không cho người khác truy cập vào 1 id nào đó

## b) Install

* npm i cookie-parser --save
* var cookieParser **=** require('cookie-parser')
* app.use(cookieParser('jdaijdj9031230'))

# 16) Evironment Variable

## a) Define

## b) Use

* console.log(process.env) để e những biến evironment hiện có
* process.env.key\_value để lấy giá trị của key value trong env
* Sử dụng module **dotenv** để lưu giá trị vào file .env
* Cài đặt:
  + npm i dotenv --save
  + require('dotenv').config()

# 17) File upload

## a) Define

* **Những kiểu encoding trong trong form**

When you make a POST request, you have to encode the data that forms the body of the request in some way.

HTML forms provide three methods of encoding.

application/x-www-form-urlencoded (the default)

multipart/form-data

text/plain

Work was being done on adding [application/json](http://www.w3.org/TR/html-json-forms/), but that has been abandoned.

(Other encodings are possible with HTTP requests generated using other means than an HTML form submission.)

The specifics of the formats don't matter to most developers. The important points are:

Never use text/plain.

When you are writing client-side code:

use multipart/form-data when your form includes any <input type="file"> elements

otherwise you can use multipart/form-data or application/x-www-form-urlencoded but application/x-www-form-urlencoded will be more efficient

* Sử dụng thư viện multer để encoding file upload

## b) Install

* $ npm install --save multer
* var multer = require('multer')
* Thêm middileware này vào route trước khi lưu dữ liệu vào database *upload.single('avatar')*