TheNetNinja – Node.js and Express

To execute .js files (without including file extension):

* node <filename>

# Basics

## Global object

* **global.setTimeOut(function, time):** set execution time out
* **global.setInterval(function, time**): set execution interval
* **\_\_dirname**: absolute path to the folder that the file resided in
* **\_\_filename**: absolute path to the file

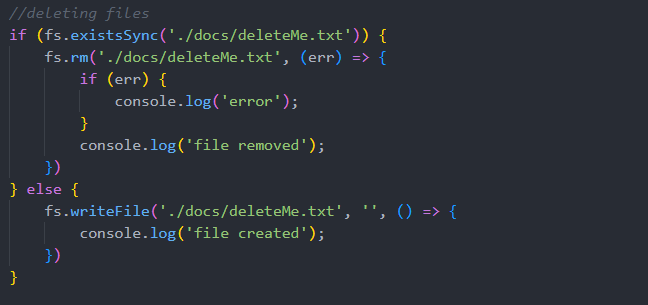
## Module and require

* **require(‘<file path>’):** find the .js file (defined a location, normally in the same directory ./) and run it, cannot access the variables inside the file by default
* to enable access to the file, need to **export** the variables: **module.export = variable**
  + The variable can be string, num, array or even object (useful when you want to access multiple variables in the file)

## The file system

Fs core module

* Read files: **Fs.readFile(<relative path>, callbackFunction(err, data))**
  + Relative path to file
  + Callback function to execute after the read is done
* Write files: **Fs.readFile(relative path, content to write, callbackFunction(err))**
  + If file does not exist, create it and write content
* Directories: fs.mkdir(relative path, callbackFunction)



## Stream and Buffer

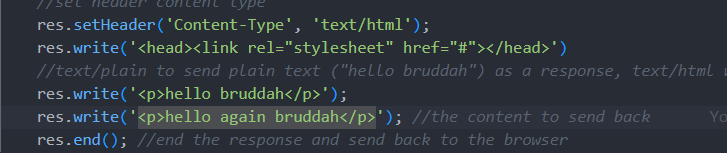
* Stream: start using data, before it has finished loading (e.g. Youtube videos)

# Server request and response



Graphical user interface, text, application

Description automatically generated



## Basic Routing: route the path to the correct html page

Text

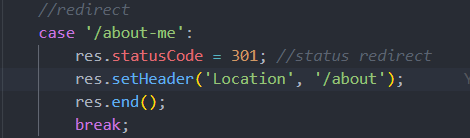
Description automatically generated

## Status code

Text

Description automatically generated

## Redirect



# NPM

Install package globally (for every projects): npm install -g

* Nodemon

Npm init

* Package.json (manage package for the project - global)
* Package-lock.json (specifically for one project - local)

Npm -I <package>

Dependencies

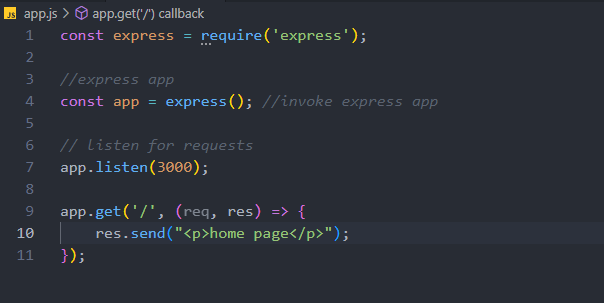
* Don’t include node\_modules in git: .gitignore node\_modules
* Npm install to install modules again

# Express Apps

Npm install express

Create express app: app.js

* Res.send(): infer type of content and status code to the browser (content type header)



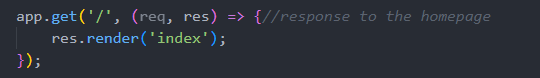
Routing and html pages

* Res.sendFile(): look for ABSOLUTE path from the computer (\_\_dirname, ./path)
* <nav>: navigation HTML
* Res.redirect(path to file)
* App.use(): response to every requests, suitable for detecting url mismatch (404 page not found)
* App.status(): set status code

# View/Template Engines – Dynamic webpages

Npm install ejs

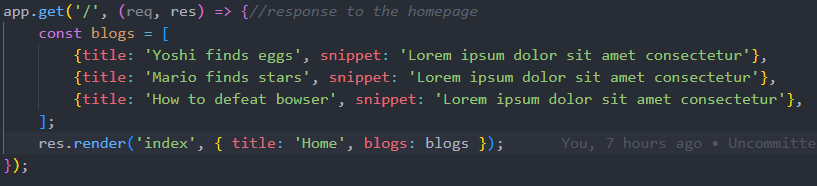
* Res.render(): render a ejs file

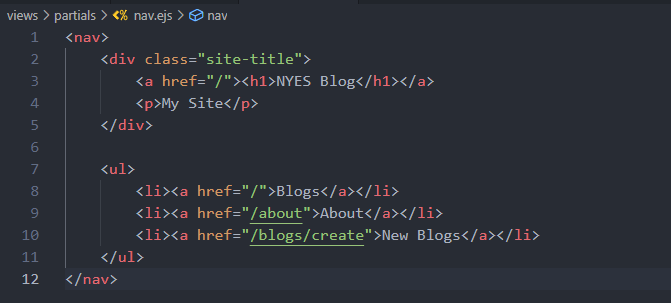


Passing data into views

* Not-so-good way
  + Graphical user interface, text

    Description automatically generated assign variables
  +  output variables
* Dynamic title: another parameter into render()
  + A screenshot of a computer

    Description automatically generated with medium confidence
  + 
  + 
* Embedded JS code into ejs: Pass variables into ejs file and use it
  + 
  + Text

    Description automatically generated
* Partial template: Inheritance of same HTML code (navigation bar e.g.) – HTML template
  + Create partial folder in views and create some files
  + “Include” those partials into ejs views
  + 
  + Text

    Description automatically generated
* Adding CSS

# Middleware - .use()

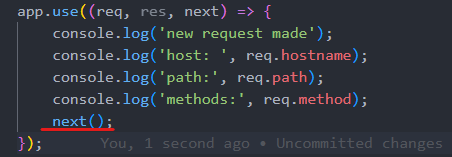
“Code that run between getting a request and sending a response”

A screenshot of a computer

Description automatically generated with medium confidence

Using next()

* Tell it to move on to the next one (doesn’t send any response)



3rd party middleware: logging, security functions, etc.

* Morgan: logging middleware



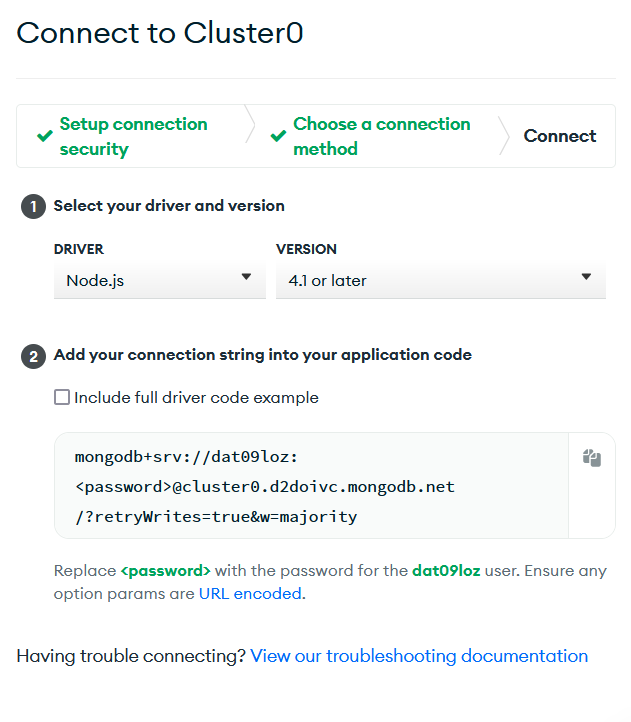
Static files

* To allow access to files, should specify which file should be public
* Static middleware

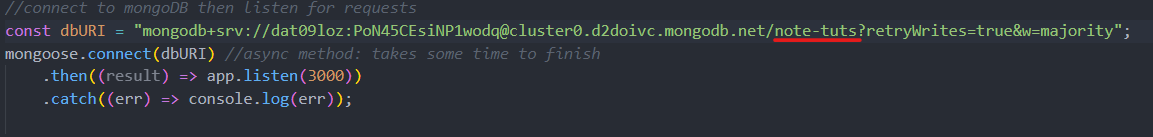
# MongoDB

Use mongoDB Atlas: dat09loz | PoN45CEsiNP1wodq

Connect to database:



Mongoose – ODM Library (remember to specify collection name)



* Blog.js

Getting and saving data: Blog

* Require Blog.js

Text

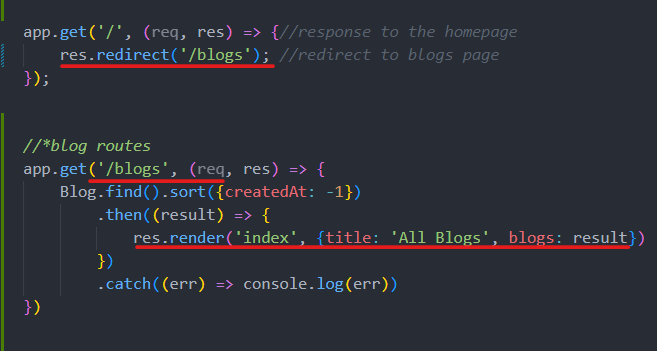
Description automatically generated

* Retrieving data

Text

Description automatically generated

Output data in views

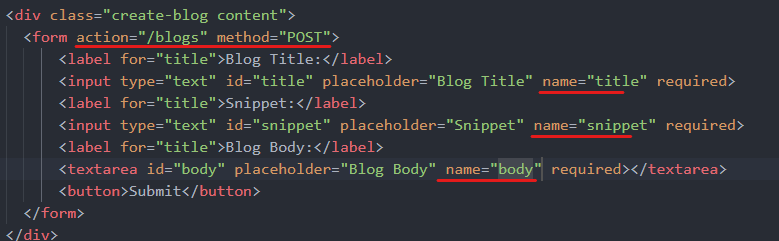


# Get, post and delete request

* Get resource
* Post request to create new data
* Delete request to delete data
* Put request for update data

Post request (for creating a new blog post):

* In .ejs file: action tag for form and name tag for each input

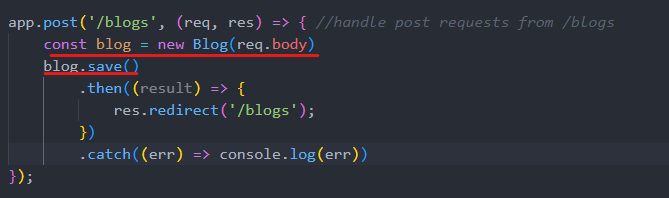


* Handle POST request on server app.js
  + Crate a middleware (process data from .ejs)





* + Create a POST req method to handle requests from /blogs



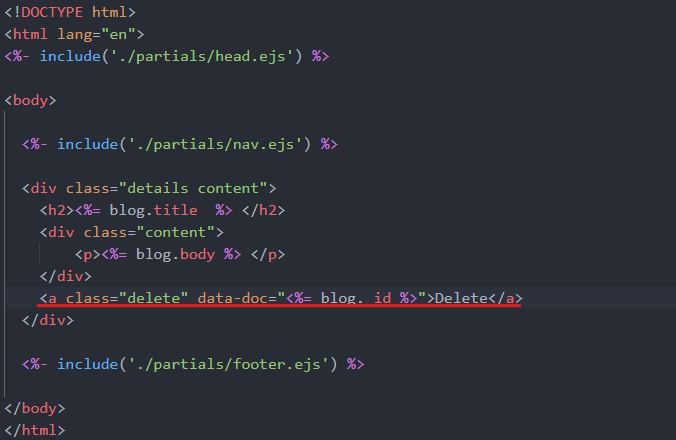
Get request (dynamic urls, for accessing each blog details in a new page)

* Surround each blog entries with its own url

A screenshot of a computer

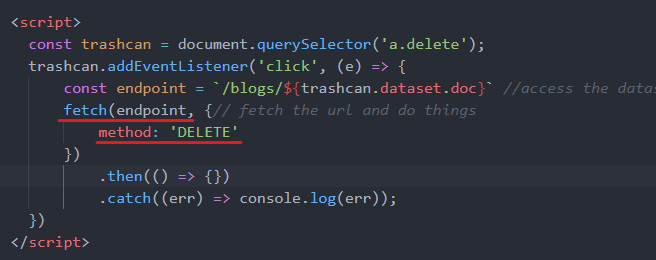
Description automatically generated with medium confidence

* Handle get request
  + Wrapped parameter in .get url ‘:id’ instead of ‘id’
  + Create a new ejs page to show blog details (with delete blogs function)

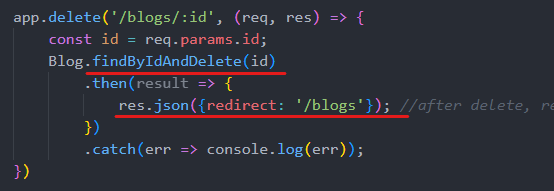


Delete request (delete each blog in their details page)

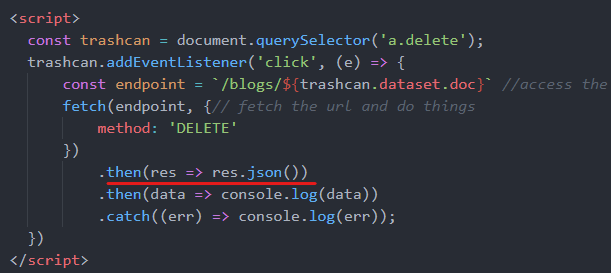
* Create a small script to handle delete – send delete request to server (details.ejs)



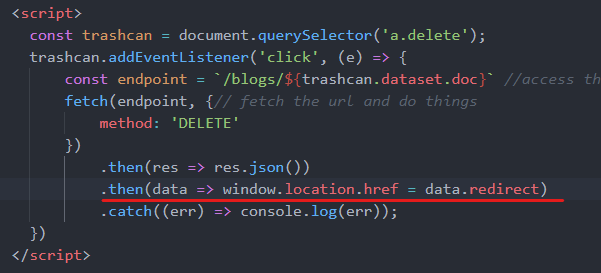
* Server delete the document based on \_id, send json as response



* Pass the response into JS object



* Set the current URL to redirect property of the data object



# MVC

Express router – handle REST operations a lot more effective

* A new router to handle blog pages: ./routes/blogRoutes.js

Text

Description automatically generated

* Export routes and import the router in app.js

Text

Description automatically generated

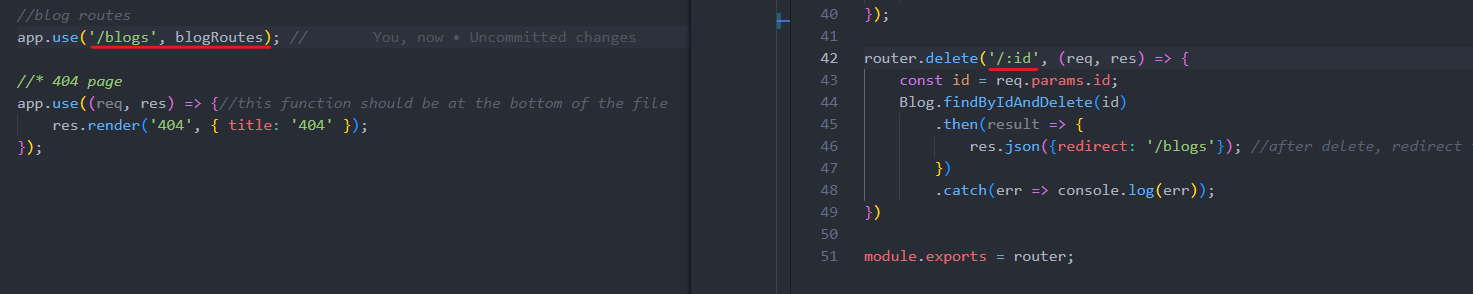
* Use the router in app.js (a middleware)

Graphical user interface, application, Teams

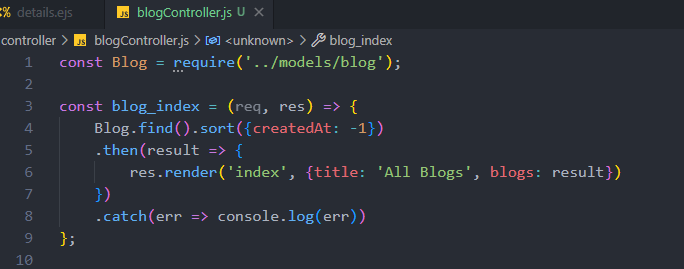
Description automatically generated

Remember to put GET request from STATIC ROUTES first (/blogs/create above /blogs/:id), or it’s not going to work

* Set /blogs to go to blogRoutes, the blogRoutes msg handler does not need to include /blogs/…. In front of it



Controller (MVC approach) – blogController.js



* In blogRoutes

Text, logo

Description automatically generated

* It is good practice to store each view into different folder (blogs view in /blogs, user views in views/user, etc.)

# Wrap Up

* Change the “delete” button to the trash icon
* Display 404 page when trying to find wrong blog \_id

