

## 20. Installing Express.js

Wednesday, May 29, 2024 11:33 AM

app.js

### Remove routes

```
Node-Max > JS app.js > ...
1 // import a global module named http
2 const http = require('http');
3
4 // import private modules
5 const routes = require('./routes');
6
7 // execute functions stored in ./routes
8 // for incoming requests
9 const server = http.createServer(routes.handler);
10
11 console.log(`routes.someText: ${routes.someText}`);
12
13 const port = 3005;
14 server.listen(port, () => {
15   console.log(`Server running at http://localhost:${port}`);
16 });
```

### Also remove routes.handler from http.createServer();

```
Node-Max > JS app.js > [e] server
1 // import a global module named http
2 const http = require('http');
3
4 // execute functions stored in ./routes
5 // for incoming requests
6 const server = http.createServer(routes.handler);
7
8 console.log(`routes.someText: ${routes.someText}`);
9
```

```
10 const port = 3005;
11 server.listen(port, () => {
12   console.log(`Server running at http://localhost:${port}`);
13 });
```

### Install Express.js as a Prod dependency

npm install --save express;

```
$ npm install --save express;
```

```
added 70 packages, and audited 102 packages in 29s
```

```
17 packages are looking for funding
  run `npm fund` for details
```

```
found 0 vulnerabilities
```

### We see "express" is now under dependencies

In fact, nodemon should be under "devDependencies" category,  
I'm just too lazy to fix this nodemon category here ;)



**Large numbers of logics** are implemented using `const app`

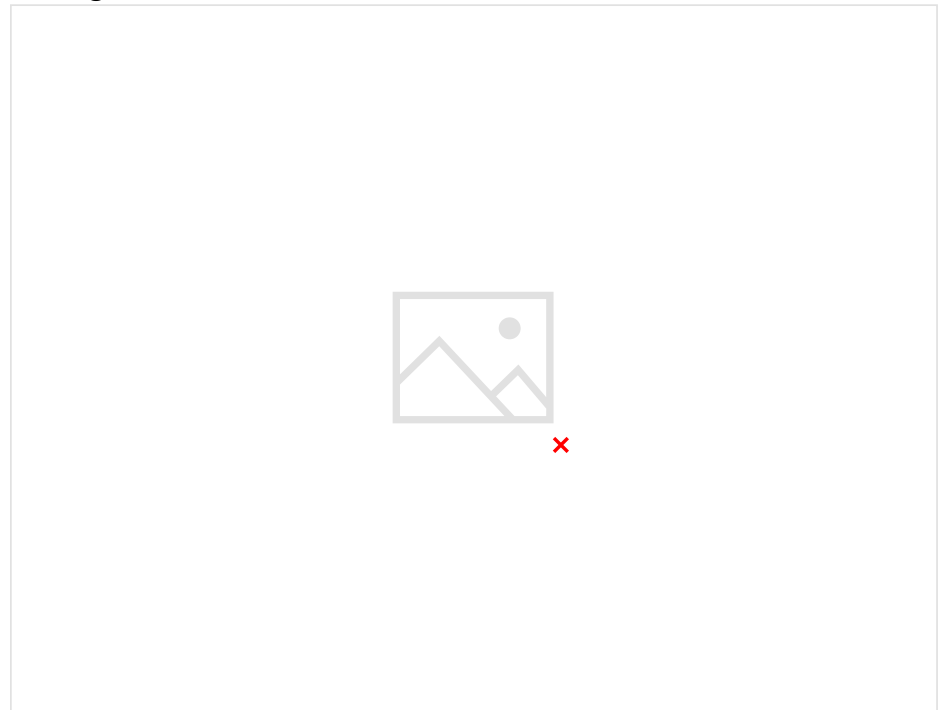


**This will create an Express server**





## Adding Middleware



## Using Express Middleware



## Express middleware design pattern



```
app.use((req, res, next) => {});
```

req = HTTP request

res = HTTP response

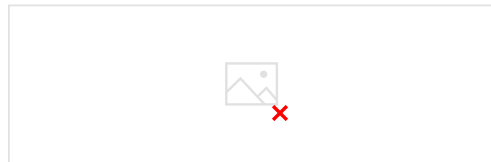
**next = next function to allow the request to travel on the next middleware**

We have 2 Middleware now

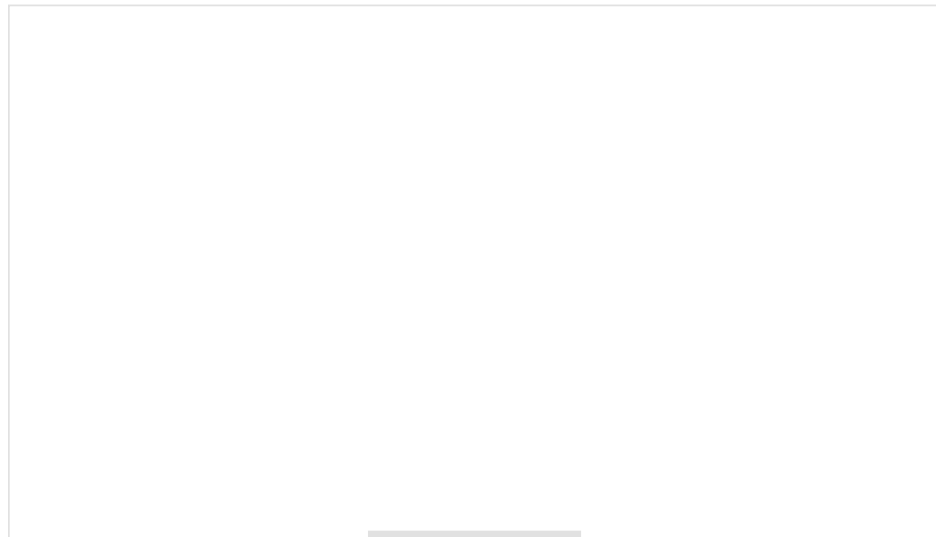




Nevertheless, we do NOT see 2 middleware in our terminal



Using `next()` in our 1st Express Middleware function  
to allow our 1st Express Middleware to travel to the next  
Middleware





**If we do NOT use next() in a Middleware, we MUST send back a response  
Otherwise, our Middleware will get stuck & entire Express app will crash**



**Let's send a HTTP response in our 2nd Middleware**



## Testing Express app in Browser







F12 Network Tab



### Looking behind the Scenes of Express.js

Express github

express/lib/response.js



express.js response.js module checks for the data type of data  
chunks





instead of

```
const server = http.createServer(app);
```

`server.listen(port);`



In fact, Express already handled this in `application.js` of Express Module

`const server = http.createServer();`



we can use **`app.listen()`**





x

## Handling different Routes

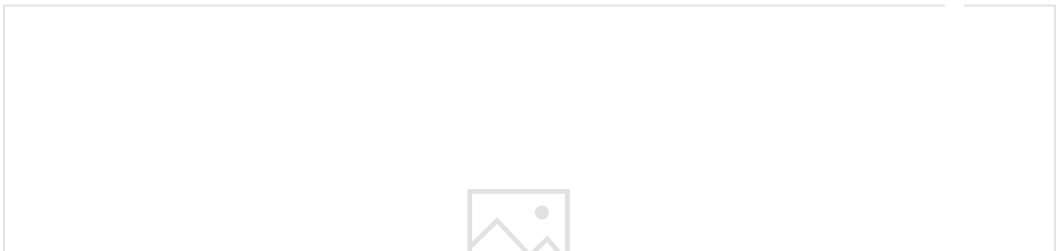
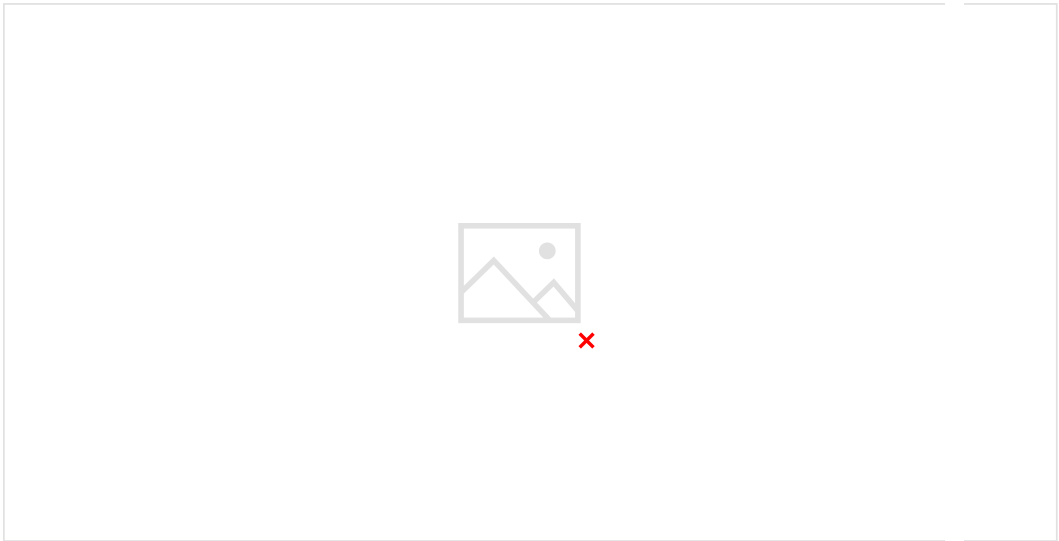
See Express documentation



x



x









×



x

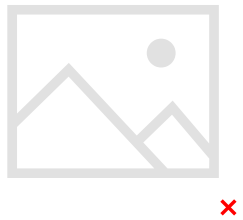


x



x

Adding an Express route for '/'



It will be inappropriate to add  
`res.send()`  
`next()`

...try

Because a middleware ends with a Response & should no longer do other actions



x

We should ensure that responses are NOT sent multiple times

inside an Express middleware

The appropriate way of using an Express middleware without sending back a Response



x

