

## 5. Working With Express.js

### \* Chapter 56: Module Introduction

---



#### What's In This Module?

What is Express.js?

Using Middleware

Working with Requests & Responses  
(Elegantly!)

Routing

Returning HTML Pages (Files)

Academind

### \* Chapter 57: What Is Express.js?

---

## What and Why?

Server Logic is Complex!

**Framework:** Helper functions,  
tools & rules that help you build  
your application!

You want to focus on your Business Logic, not  
on the nitty-gritty Details!

Use a Framework for the Heavy Lifting!

express

## Alternatives to Express.js

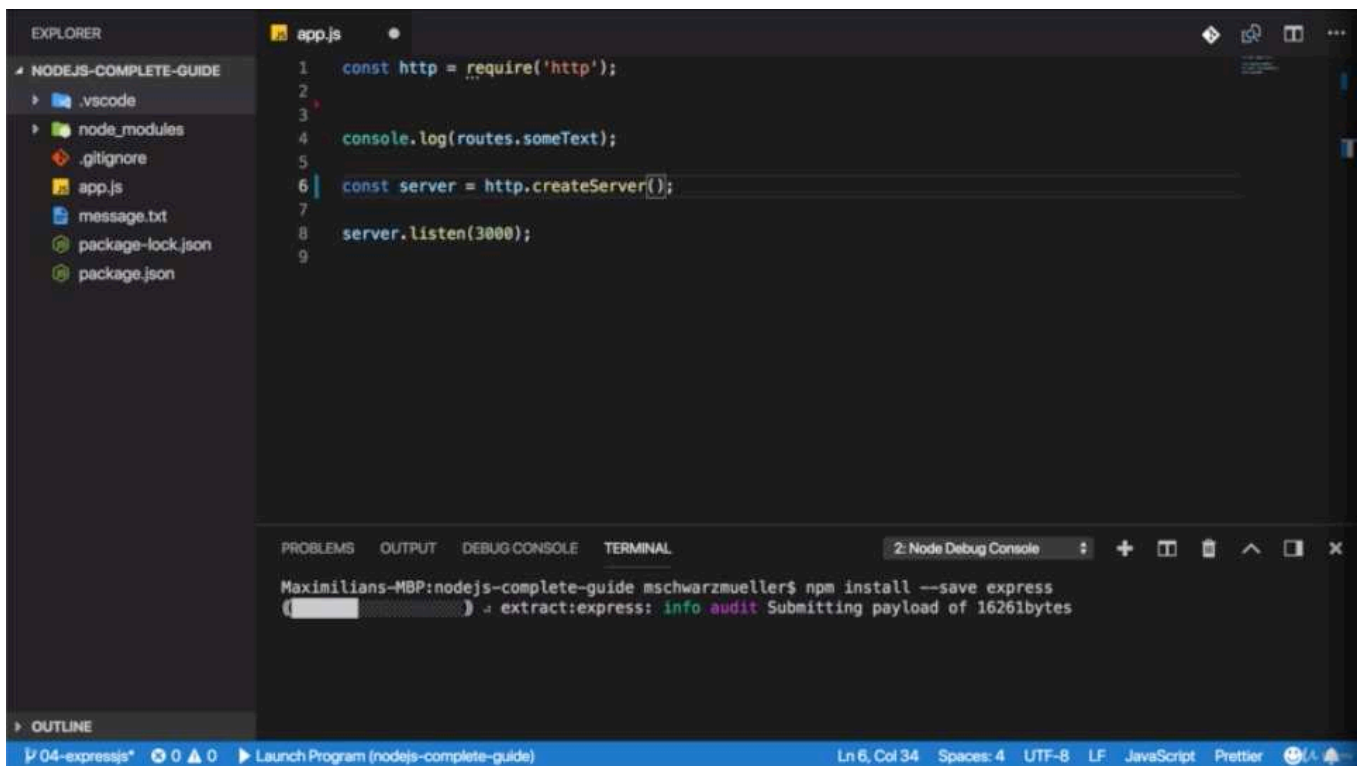
Vanilla Node.js

Adonis.js

Koa

Sails.js

...



The screenshot shows a VS Code editor with a file explorer on the left containing files like .vscode, node\_modules, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor displays `app.js` with the following code:

```
1 const http = require('http');
2
3
4 console.log(routes.someText);
5
6 const server = http.createServer();
7
8 server.listen(3000);
9
```

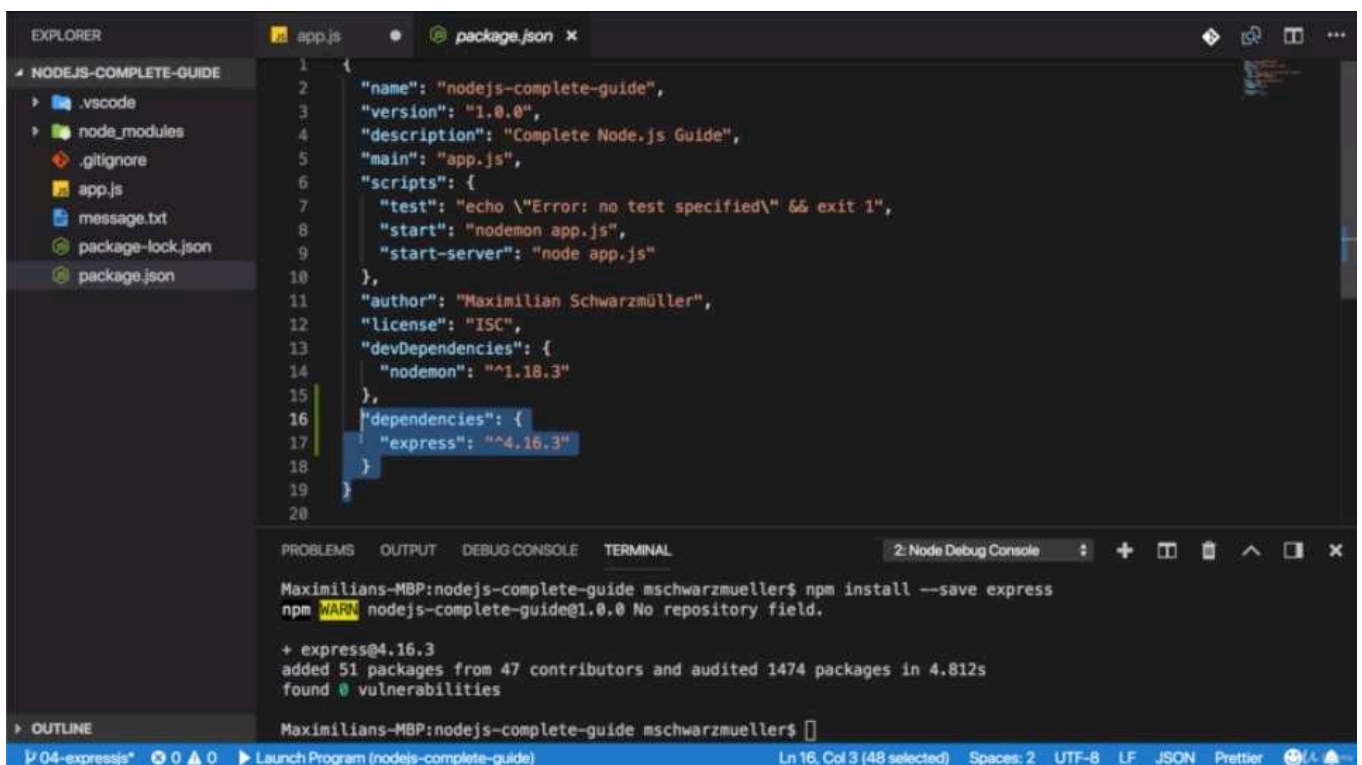
The bottom panel shows the **TERMINAL** tab with the command `npm install --save express` and its output:

```
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm install --save express
( [REDACTED] ) - extract:express: info audit Submitting payload of 16261bytes
```

## \* Chapter 58: Installing Express.js

1. update

- app.js



The screenshot shows a VS Code editor with the `package.json` file open. The file content is as follows:

```
1 {
2   "name": "nodejs-complete-guide",
3   "version": "1.0.0",
4   "description": "Complete Node.js Guide",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1",
8     "start": "nodemon app.js",
9     "start-server": "node app.js"
10  },
11   "author": "Maximilian Schwarzmüller",
12   "license": "ISC",
13   "devDependencies": {
14     "nodemon": "^1.18.3"
15  },
16   "dependencies": {
17     "express": "^4.16.3"
18  }
19 }
```

The bottom panel shows the **TERMINAL** tab with the command `npm install --save express` and its output:

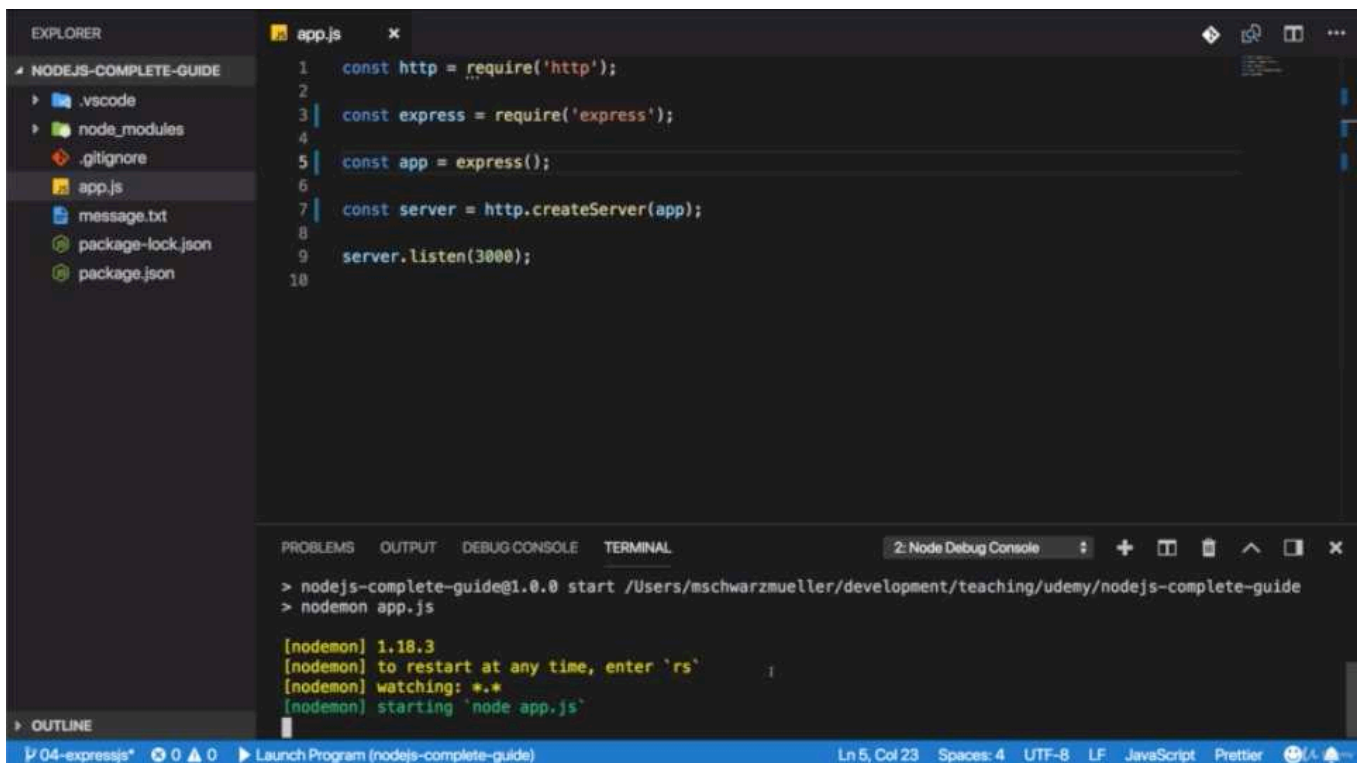
```
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm install --save express
npm WARN nodejs-complete-guide@1.0.0 No repository field.

+ express@4.16.3
added 51 packages from 47 contributors and audited 1474 packages in 4.812s
found 0 vulnerabilities

Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
```

- why not `--save-dev` but `--save`? because this will be a production dependency. we don't just use that as a tool during development. it will be an integral part of the application we ship and therefore it definitely also has to be installed on any server or any computer where

we run our application once we deploy it. it's a major piece of our application.



The screenshot shows a VS Code editor with a file explorer on the left showing the project structure: NODEJS-COMPLETE-GUIDE, .vscode, node\_modules, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor displays the app.js file with the following code:

```
1 const http = require('http');
2
3 const express = require('express');
4
5 const app = express();
6
7 const server = http.createServer(app);
8
9 server.listen(3000);
10
```

The terminal at the bottom shows the command to start the application and the output from nodemon:

```
> nodejs-complete-guide@1.0.0 start /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide
> nodemon app.js

[nodemon] 1.18.3
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching: *.*
[nodemon] starting 'node app.js'
```

- after npm start, you will actually have a running server.

```
//app.js

const http = require('http');

/**this will initialize new object
*/
const express = require('express');

const app = express()

/** 'app' happens to be a valid request handler
 * so you can pass 'app' here to create server.
 *
 * app sets up a certain way of handling incoming requests
 * that defines or that is a key characteristic of express.js
 */
const server = http.createServer(app);

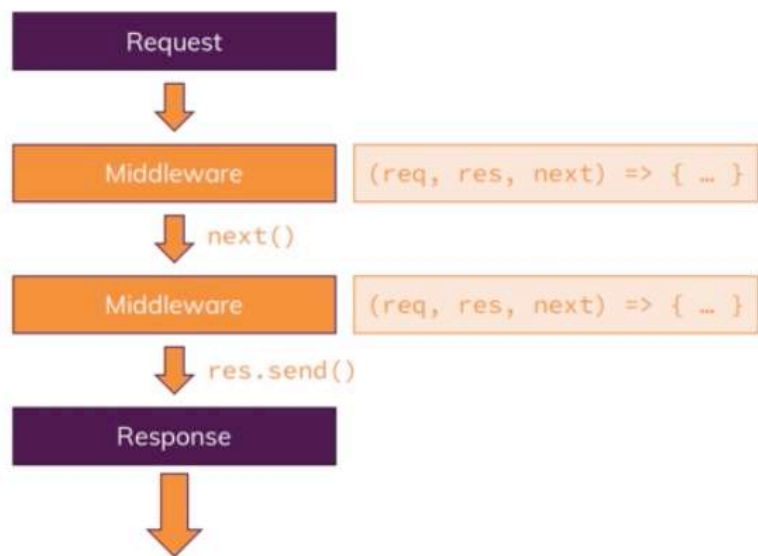
server.listen(3000);
```

## \* Chapter 59: Adding Middleware

### 1. update

- app.js

## It's all about Middleware

Udacity

- Express.js is all about middleware which means an incoming request is automatically funneled through a bunch of function by express.js
- so instead of just having one request handler, you will actually have a possibility of hooking in multiple

functions which the request will go through until you send a response.

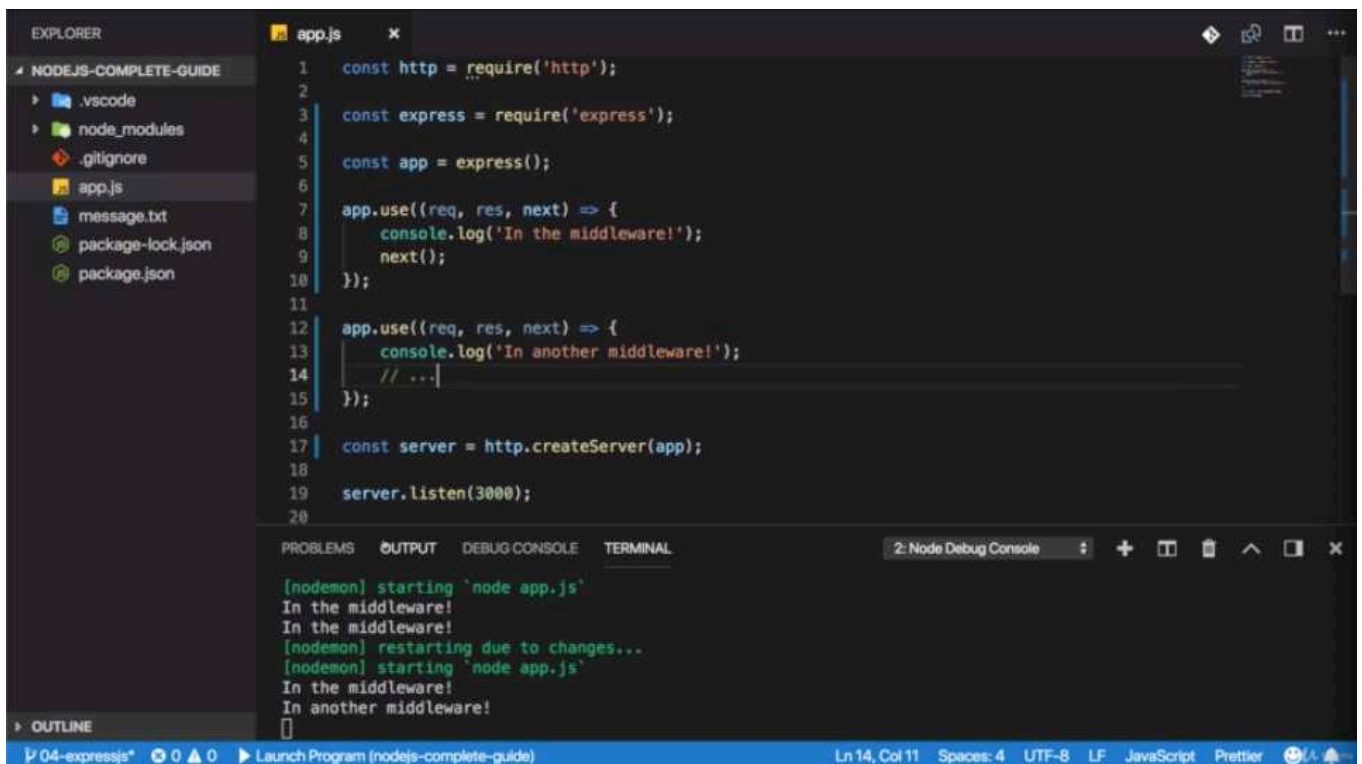
- This allows you to split your code into multiple blocks or pieces instead of having one huge function that does everything and this is the pluggable nature of express.js, where you can easily add other third party packages which simply happen to give you such middleware functions that you can plug into express.js and add certain functionalities



- this spinner will keep on spinning. so we don't get a response. because we got no logic.

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'NODEJS-COMPLETE-GUIDE' with files like '.vscode', 'node\_modules', '.gitignore', 'app.js', 'message.txt', 'package-lock.json', and 'package.json'. The main editor window displays the content of 'app.js'. The code defines an Express application with two middleware functions. The first middleware logs 'In the middleware!' and the second logs 'In another middleware!'. The server is configured to listen on port 3000. The Terminal at the bottom shows the output of running 'node app.js', which includes messages from nodemon about restarting and the first log statement 'In the middleware!'.

- but i don't see in another middleware. because we need to type 'next()' to get next middleware.



The screenshot shows a VS Code editor window with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'NODEJS-COMPLETE-GUIDE' with files like .vscode, node\_modules, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor shows the content of 'app.js', which is a Node.js application using Express.js. The code defines an Express app, adds two middleware functions, creates a server, and listens on port 3000. The terminal at the bottom shows the output of running 'node app.js', which includes messages from nodemon and the application's logs: 'In the middleware!' and 'In another middleware!'.

```
1  const http = require('http');
2
3  const express = require('express');
4
5  const app = express();
6
7  app.use((req, res, next) => {
8    console.log('In the middleware!');
9    next();
10 });
11
12 app.use((req, res, next) => {
13   console.log('In another middleware!');
14   // ...
15 });
16
17 const server = http.createServer(app);
18
19 server.listen(3000);
20
```

PROBLEMS OUTPUT DEBUG-CONSOLE TERMINAL 2: Node Debug Console

```
[nodemon] starting 'node app.js'
In the middleware!
In the middleware!
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
In the middleware!
In another middleware!
```

//app.js

```
const http = require('http');
```

```
const express = require('express');
```

```
const app = express()
```

```
/**'use()' allows us to add a new middleware function
```

```
 * use() allows us this function to be executed for every incoming request
```

```
 *
```

```
 * 'next' is a function that will be passed to this function by Express.js
```

```
 */
```

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

```
  /**this function you are receiving has to be executed
```

```
  * to allow the request to travel onto the next middleware */
```

```
  console.log('In the middleware')
```

```
  /**if we don't call 'next', we should actually send back a response
```

```
  * because otherwise the request can't continue
```

```
  * so it will never reach a place where we might send a response
```

```
  */
```

```
  next();
```

```
})
```

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

```
  console.log('In another middleware')
```

```
  //...
```

```
})
```

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

```
server.listen(3000);
```

## \* Chapter 60: How Middleware Works

1. update

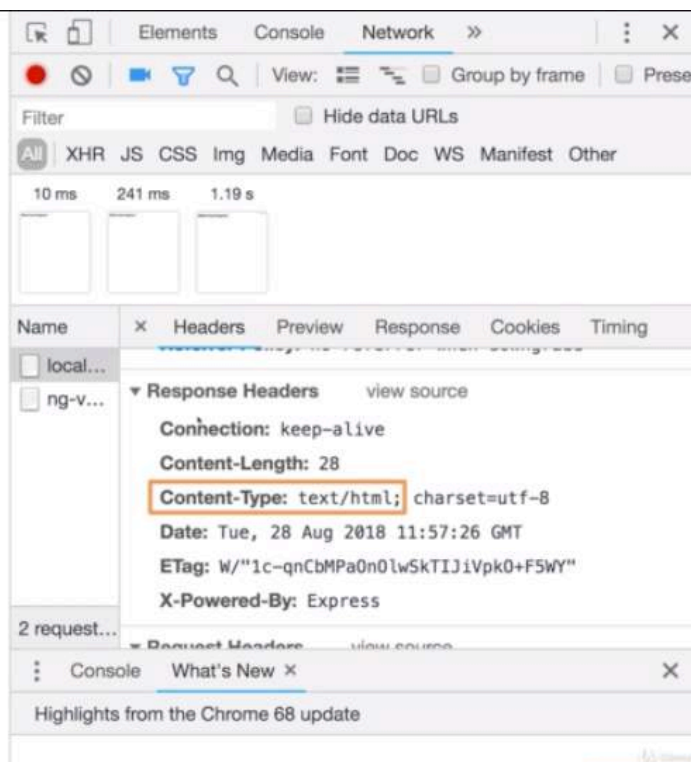
- app.js

---

**Hello from Express!**

---

**Hello from Express!**



- under header, the content-type is automatically set to text/html here. this is another feature provided by Express.



```
const http = require('http');

const express = require('express');

const app = express()

app.use((req, res, next) => {
  console.log('In the middleware')
  next();
})

app.use((req, res, next) => {
  console.log('In another middleware')
  /**'send()' allows us to send well a response
   * this allows us to attach a body which is of type any.
   *
   * this is particularly easier
   * once we start sending back real files or the content of files
   */
  res.send('<h1>Hello from Express!</h1>')
})

const server = http.createServer(app);

server.listen(3000);
```

## 🔗 \* Chapter 61: Express.js - Looking Behind The Scenes

---

### 1. update

- app.js

```

79 *   });
80 *
81 * @param {Object} links
82 * @return {ServerResponse}
83 * @public
84 */
85
86 res.links = function(links){
87   var link = this.get('Link') || '';
88   if (!link) link = '';
89   return this.set('Link', link + Object.keys(links).map(function(rel){
90     return '<' + links[rel] + '>; rel="' + rel + '"';
91   }).join(', '));
92 };
93
94 /**
95 * Send a response.
96 *
97 * Examples:
98 *
99 *   res.send(Buffer.from('wahoo'));
100 *   res.send({ some: 'json' });
101 *   res.send('<p>some html</p>');
102 *
103 * @param {string|number|boolean|object|Buffer} body
104 * @public
105 */
106
107 res.send = function send(body) {
108   var chunk = body;
109   var encoding;
110   var req = this.req;
111   var type;
112
113   // settings
114   var app = this.app;
115
116   // allow status / body
117   if (arguments.length === 2) {
118     // res.send(body, status) backwards compat
119     if (typeof arguments[0] !== 'number' && typeof arguments[1] !== 'number') {

```

- you will see how the send function.

The screenshot shows a VS Code editor with a file named `app.js` open. The file contains the following code:

```

1  const http = require('http');
2
3  const express = require('express');
4
5  const app = express();
6
7  app.use((req, res, next) => {
8    console.log('In the middleware!');
9    next(); // Allows the request to continue to the next middleware in line
10 });
11
12 app.use((req, res, next) => {
13   console.log('In another middleware!');
14   res.send('<h1>Hello from Express!</h1>');
15 });
16
17 const server = http.createServer(app);
18
19 server.listen(3000);
20

```

The terminal at the bottom shows the output of running the application:

```

[nodemon] starting 'node app.js'
In the middleware!
In another middleware!
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
In the middleware!
In another middleware!

```

- so basically a function we are calling here, how this is defined internally and this helps us understand
- by the way, this is always a great technique if you wanna see what something does behind the scene and if you need to do something yourself, set someHeader or if that is done for you and we had that default header of text/html. so let's see what send does internally.

```

129 // disambiguate res.send(status) and res.send(status, num)
130 if (typeof chunk === 'number' && arguments.length === 1) {
131   // res.send(status) will set status message as text string
132   if (!this.get('Content-Type')) {
133     this.type('txt');
134   }
135 }
136
137 deprecate('res.send(status): Use res.sendStatus(status) instead');
138 this.statusCode = chunk;
139 chunk = statuses(chunk)
140 }
141
142 switch (typeof chunk) {
143   // string defaulting to html
144   case 'string':
145     if (!this.get('Content-Type')) {
146       this.type('html');
147     }
148     break;
149   case 'boolean':
150   case 'number':
151   case 'object':
152     if (chunk === null) {
153       chunk = '';
154     } else if (Buffer.isBuffer(chunk)) {
155       if (!this.get('Content-Type')) {
156         this.type('bin');
157       }
158     } else {
159       return this.json(chunk);
160     }
161     break;
162 }
163
164 // write strings in utf-8
165 if (typeof chunk === 'string') {
166   encoding = 'utf8';
167   type = this.get('Content-Type');
168   // reflect this in content-type

```

- it basically analyzes what kind of data you sending.

The screenshot shows a VS Code editor with a file explorer on the left containing files like .vscode, node\_modules, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor displays the content of app.js, which defines an Express application with two middleware functions and a server listener. The first middleware logs 'In the middleware!' and calls next(). The second middleware logs 'In another middleware!', sends an HTML response, and calls next(). The server is created and listens on port 3000. The terminal at the bottom shows the output of running the application, confirming the middleware execution order.

```

1  const http = require('http');
2
3  const express = require('express');
4
5  const app = express();
6
7  app.use((req, res, next) => {
8    console.log('In the middleware!');
9    next(); // Allows the request to continue to the next middleware in line
10 });
11
12 app.use((req, res, next) => {
13   console.log('In another middleware!');
14   res.send('<h1>Hello, from Express!</h1>');
15 });
16
17 const server = http.createServer(app);
18
19 server.listen(3000);
20

```

```

[nodemon] starting 'node app.js'
In the middleware!
In another middleware!
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
In the middleware!
In another middleware!

```

- if it's a string data, so some text as we are doing it here, in this case, it sets content-type to html. but only if we haven't set it yet. so it checks if the content type header is not present yet in which case it sets it, otherwise it would leave our default.
- if we have other values like a number, a boolean and so on, it would set it to binary or json data.

EXPLORER

- NODEJS-COMPLETE-GUIDE
  - .vscode
  - node\_modules
  - .gitignore
  - app.js
  - message.txt
  - package-lock.json
  - package.json

```
11
12 app.use((req, res, next) => {
13   console.log('In another middleware!');
14   res.send('<h1>Hello from Express!</h1>');
15 });
16
17 const server = http.createServer(app);
18
19 server.listen(3000);
20
```

PROBLEMS OUTPUT DEBUG-CONSOLE TERMINAL

2: Node Debug Console

```
[nodemon] starting `node app.js`
In the middleware!
In another middleware!
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
In the middleware!
In another middleware!
```

04-expressjs 0 0 Launch Program (nodejs-complete-guide) Ln 17, Col 1 (60 selected) Spaces: 4 UTF-8 LF JavaScript Prettier

EXPLORER

- NODEJS-COMPLETE-GUIDE
  - .vscode
  - node\_modules
  - .gitignore
  - app.js
  - message.txt
  - package-lock.json
  - package.json

```
7 app.use((req, res, next) => {
8   console.log('In the middleware!');
9   next(); // Allows the request to continue to the next middleware in line
10 });
11
12 app.use((req, res, next) => {
13   console.log('In another middleware!');
14   res.send('<h1>Hello from Express!</h1>');
15 });
16
17 app.listen(3000);
18
```

PROBLEMS OUTPUT DEBUG-CONSOLE TERMINAL

2: Node Debug Console

```
[nodemon] starting `node app.js`
In the middleware!
In another middleware!
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
In the middleware!
In another middleware!
```

04-expressjs 0 0 Launch Program (nodejs-complete-guide) Ln 17, Col 18 Spaces: 4 UTF-8 LF JavaScript Prettier

- now one other interesting thing to see is that we can actually also shorten this code here where we set up the server.
- `app.listen()` do both these things for us, something we can see in the official code.

```

592     tryRender(view, renderOptions, done);
593   };
594
595   /**
596    * Listen for connections.
597    *
598    * A node "http.Server" is returned, with this
599    * application (which is a "Function") as its
600    * callback. If you wish to create both an HTTP
601    * and HTTPS server you may do so with the "http"
602    * and "https" modules as shown here:
603    *
604    *   var http = require('http')
605    *       , https = require('https')
606    *       , express = require('express')
607    *       , app = express();
608    *
609    *   http.createServer(app).listen(80);
610    *   https.createServer( { ... }, app).listen(443);
611    *
612    * @return {http.Server}
613    * @public
614    */
615
616   app.listen = function listen() {
617     var server = http.createServer(this);
618     return server.listen.apply(server, arguments);
619   };
620
621   /**
622    * Log error using console.error.
623    *
624    * @param {Error} err
625    * @private
626    */
627
628   function logerror(err) {
629     /* istanbul ignore next */
630     if (this.get('env') !== 'test') console.error(err.stack || err.toString());
631   }

```

- listen function in the end does the 2 things we did before.

```

//app.js

const http = require('http');

const express = require('express');

const app = express()

app.use((req, res, next) => {
  console.log('In the middleware')
  next();
})

app.use((req, res, next) => {
  console.log('In another middleware')
  res.send('<h1>Hello from Express!</h1>')
})

app.listen(3000);

```

## \* Chapter 62: Handling Different Routes

### 1. update

- app.js

Hello from Express!

Elements Console Network >>

View: [Icons] Group by frame [X] Preserve [X]

Filter [ ] Hide data URLs

[All] XHR JS CSS Img Media Font Doc WS Manifest Other

10 ms 196 ms

Name	Status	Type	...	Size	...	Waterfall	▲
localhost	200	do...	...	233 B	...	[Waterfall]	
ng-validate.js	200	scr...	...	(from dis...)	...	[Waterfall]	

2 requests | 233 B transferred | Finish: 229 ms | DOMContentLoaded: 136 m...

⋮ Console What's New ✕

Highlights from the Chrome 68 update

Chrome File Edit View History Bookmarks People Window Help

localhost:3000/ add-product

View: [Icons] Group by frame [X] Preserve [X]

Filter [ ] Hide data URLs

[All] XHR JS CSS Img Media Font Doc WS Manifest Other

Hit ⌘ R to reload and capture filmstrip.

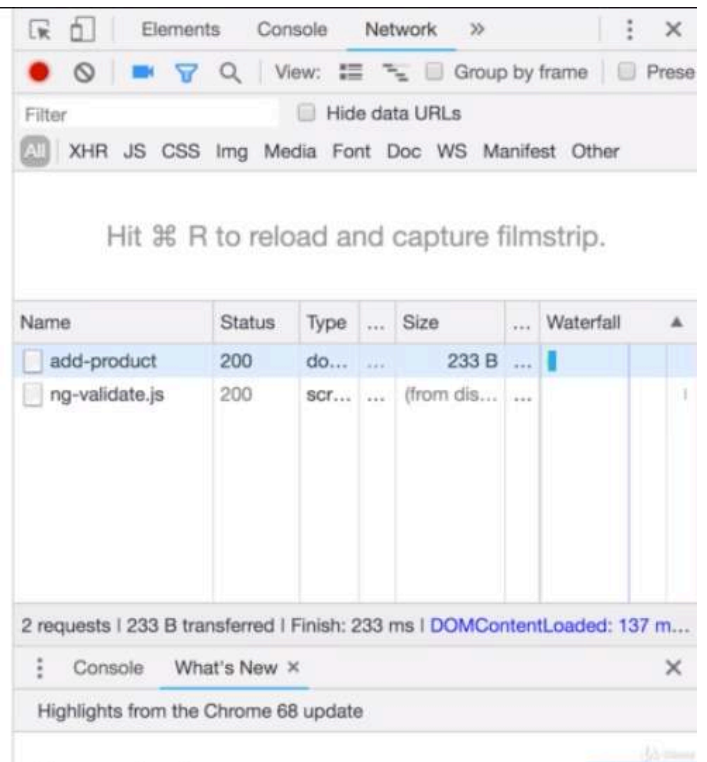
Name	Status	Type	...	Size	...	Waterfall	▲
add-product	200	do...	...	233 B	...	[Waterfall]	
ng-validate.js	200	scr...	...	(from dis...)	...	[Waterfall]	

2 requests | 233 B transferred | Finish: 233 ms | DOMContentLoaded: 137 m...

⋮ Console What's New ✕

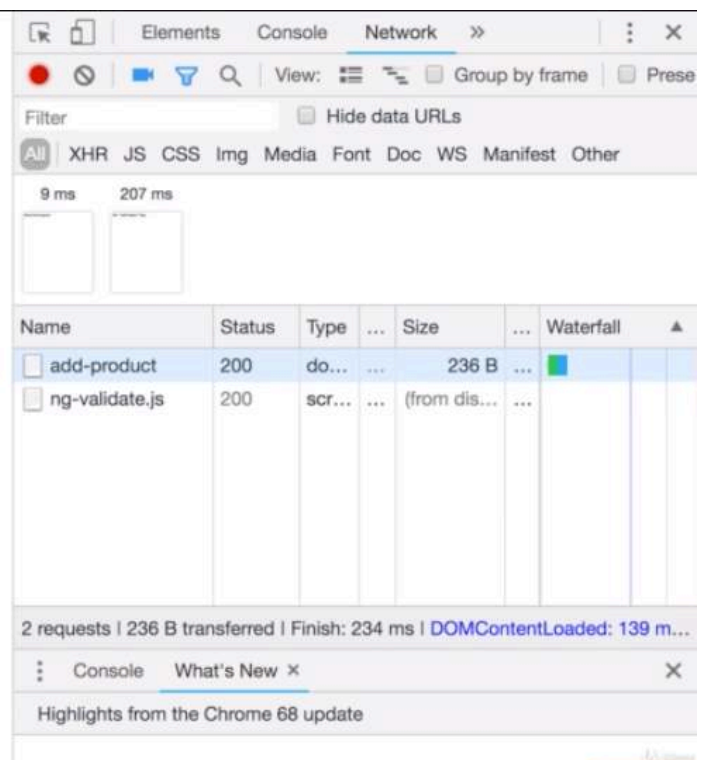
Highlights from the Chrome 68 update

## Hello from Express!



- if we type not '/' but '/add-product', we still see 'Hello from Express' and we still see i'm in another middleware, so this middleware gets executed for both slash and add-product because this '/' doesn't mean that the full path. so the part after the domain has to be a slash but that it has to start with that.

## The "Add Product" Page





Hello from Express!

Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	...	Size	...	Waterfall	▲
localhost	200	do...	...	233 B	...		
ng-validate.js	200	scr...	...	(from dis...	...		

2 requests | 233 B transferred | Finish: 223 ms | DOMContentLoaded: 131 m...

Console What's New ✕

Highlights from the Chrome 68 update

**EXPLORER**

- NODEJS-COMPLETE-GUIDE
- node\_modules
- app.js
- message.txt
- package-lock.json
- package.json

**app.js**

```
1 const express = require('express');
2
3 const app = express();
4
5 app.use('/', (req, res, next) => {
6   console.log('This always runs!');
7   next();
8 });
9
10 app.use('/add-product', (req, res, next) => {
11   console.log('In another middleware!');
12   res.send('<h1>The "Add Product" Page</h1>');
13 });
14
15 app.use('/', (req, res, next) => {
16   console.log('In another middleware!');
17   res.send('<h1>Hello from Express!</h1>');
18 });
19
20 app.listen(3000);
```

**TERMINAL**

```
at tryModuleLoad (internal/modules/cjs/loader.js:538:12)
at Function.Module._load (internal/modules/cjs/loader.js:530:3)
at Function.Module.runMain (internal/modules/cjs/loader.js:742:12)
at startup (internal/bootstrap/node.js:266:19)
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
```

04-expressjs\* 0 ▲ 0 ▶ Launch Program (nodejs-complete-guide) Ln 6, Col 38 Spaces: 4 UTF-8 LF JavaScript Prettier: ✓



## Hello from Express!

Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	...	Size	...	Waterfall	▲
localhost	200	do...	...	233 B	...		
ng-validate.js	200	scr...	...	(from dis...	...		

2 requests | 233 B transferred | Finish: 239 ms | DOMContentLoaded: 136 m...

Console What's New ✕

Highlights from the Chrome 68 update

## The "Add Product" Page

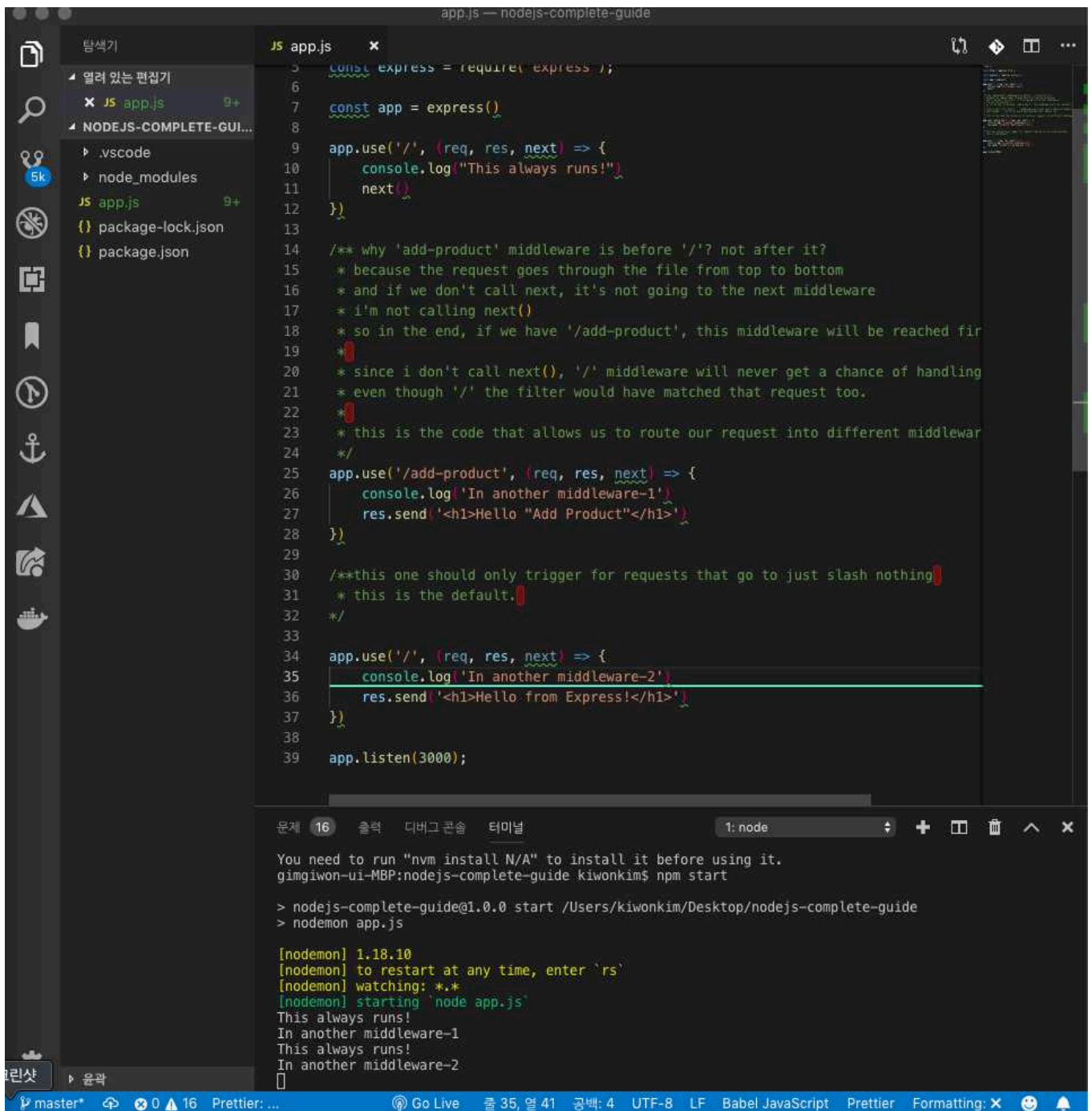
Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	...	Size	...	Waterfall	▲
add-product	200	do...	...	236 B	...		
ng-validate.js	200	scr...	...	(from dis...	...		

2 requests | 236 B transferred | Finish: 219 ms | DOMContentLoaded: 126 m...

Console What's New ✕

Highlights from the Chrome 68 update



The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'NODEJS-COMPLETE-GUI...' with files like 'package-lock.json' and 'package.json'. The main editor displays 'app.js' with the following code:

```
1  const express = require('express');
2
3  const app = express()
4
5  app.use('/', (req, res, next) => {
6    console.log('This always runs!')
7    next()
8  })
9
10 /** why 'add-product' middleware is before '/'? not after it?
11  * because the request goes through the file from top to bottom
12  * and if we don't call next, it's not going to the next middleware
13  * i'm not calling next()
14  * so in the end, if we have '/add-product', this middleware will be reached fir
15  * since i don't call next(), '/' middleware will never get a chance of handling
16  * even though '/' the filter would have matched that request too.
17  *
18  * this is the code that allows us to route our request into different middlewar
19  */
20 app.use('/add-product', (req, res, next) => {
21   console.log('In another middleware-1')
22   res.send('<h1>Hello "Add Product"</h1>')
23 })
24
25 /**this one should only trigger for requests that go to just slash nothing
26  * this is the default.
27  */
28
29 app.use('/', (req, res, next) => {
30   console.log('In another middleware-2')
31   res.send('<h1>Hello from Express!</h1>')
32 })
33
34 app.listen(3000);
```

The terminal window at the bottom shows the following output:

```
문제 16 출력 디버그 콘솔 터미널
1: node
You need to run "npm install N/A" to install it before using it.
gimgiwon-ui-MBP:nodejs-complete-guide kiwonkim$ npm start
> nodejs-complete-guide@1.0.0 start /Users/kiwonkim/Desktop/nodejs-complete-guide
> nodemon app.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
This always runs!
In another middleware-1
This always runs!
In another middleware-2
[]
```

- we have this runs twice because this first app.use() always runs.

```
//app.js
```

```
const express = require('express');
```

```
const app = express()
```

```
/** why 'add-product' middleware is before '/'? not after it?
 * because the request goes through the file from top to bottom
 * and if we don't call next, it's not going to the next middleware
 * i'm not calling next()
 * so in the end, if we have '/add-product',
 * this middleware will be reached first.
 *
 * since i don't call next(), '/' middleware will never
```

```

* get a chance of handling that request
* even though '/' the filter would have matched that request too.
*
* this is the code that allows us to route our request
* into different middleware.
*/
app.use('/add-product', (req, res, next) => {
  console.log('In another middleware-1')
  res.send('<h1>Hello "Add Product"</h1>')
})

/**this one should only trigger for requests
 * that go to just slash nothing
 * this is the default.
 */

app.use('/', (req, res, next) => {
  console.log('In another middleware-2')
  res.send('<h1>Hello from Express!</h1>')
})

app.listen(3000);

```

## 🔗 \* Assignment: Time To Practice - Express.js

```

//app.js

const express = require('express');

const app = express();

// app.use((req, res, next) => {
//   console.log('First Middleware');
//   next();
// });

// app.use((req, res, next) => {
//   console.log('Second Middleware');
//   res.send('<p>Assignment solved (almost!)</p>');
// });

app.use('/users', (req, res, next) => {
  console.log('/users middleware');
  res.send(
    '<p>The Middleware that handles just /users</p>'
  );
});

app.use('/', (req, res, next) => {
  console.log('/ middleware');
  res.send('<p>The Middleware that handles just /</p>');
});

```

```
});
```

```
app.listen(3000);
```

```
//package.json
```

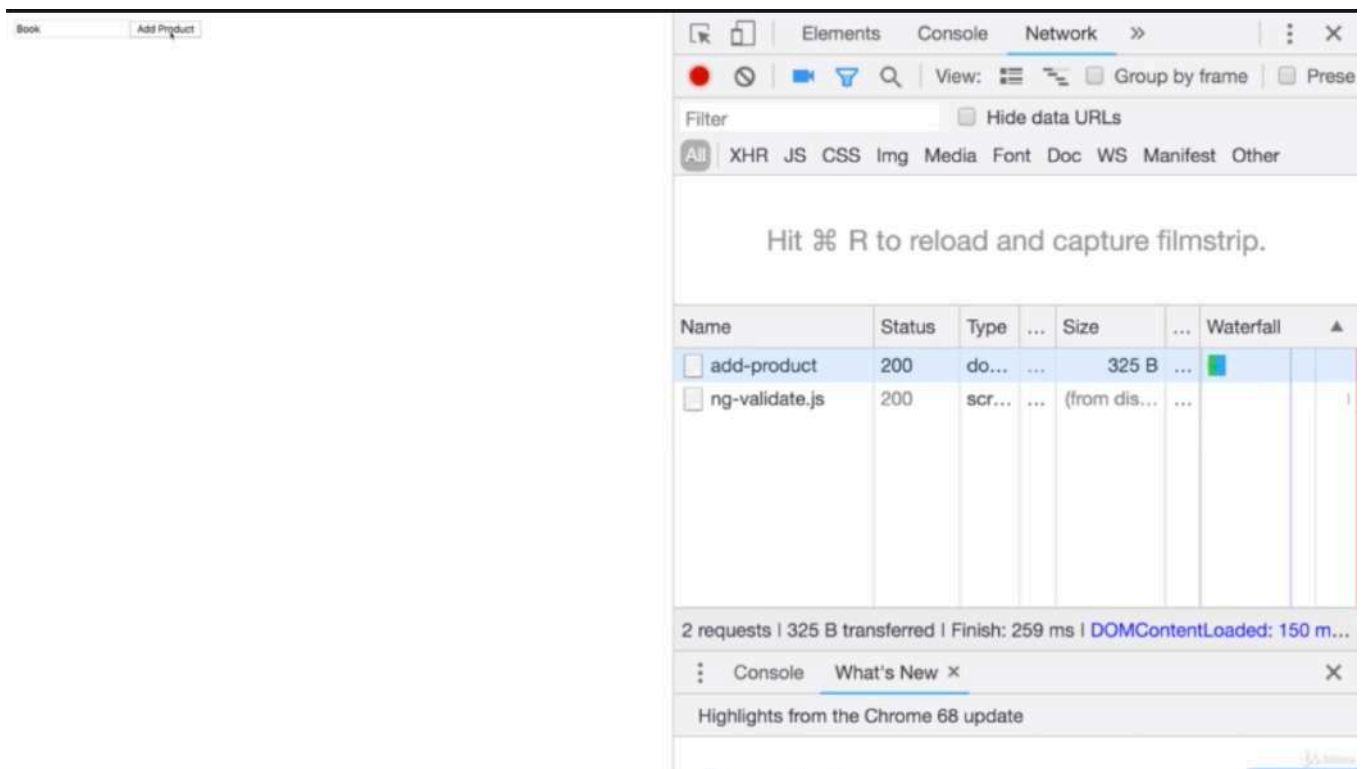
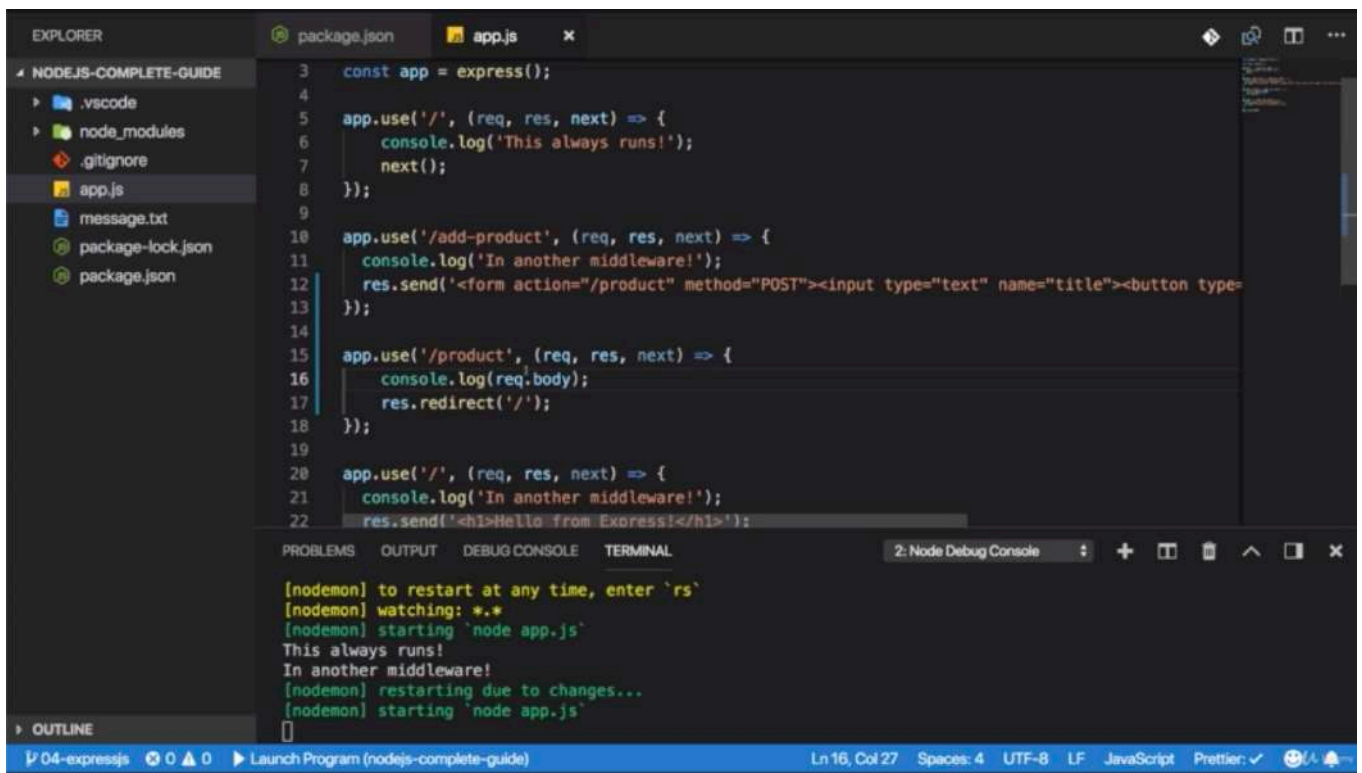
```
{
  "name": "nodejs-complete-guide",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.3"
  },
  "devDependencies": {
    "nodemon": "^1.18.3"
  }
}
```

## 🔗 \* Chapter 63: Parsing Incoming Requests

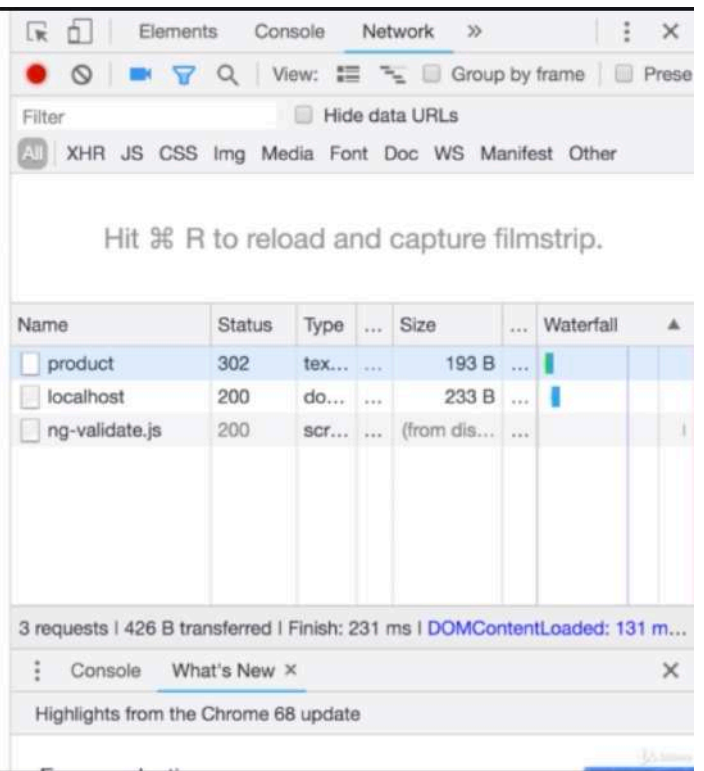
---

### 1. update

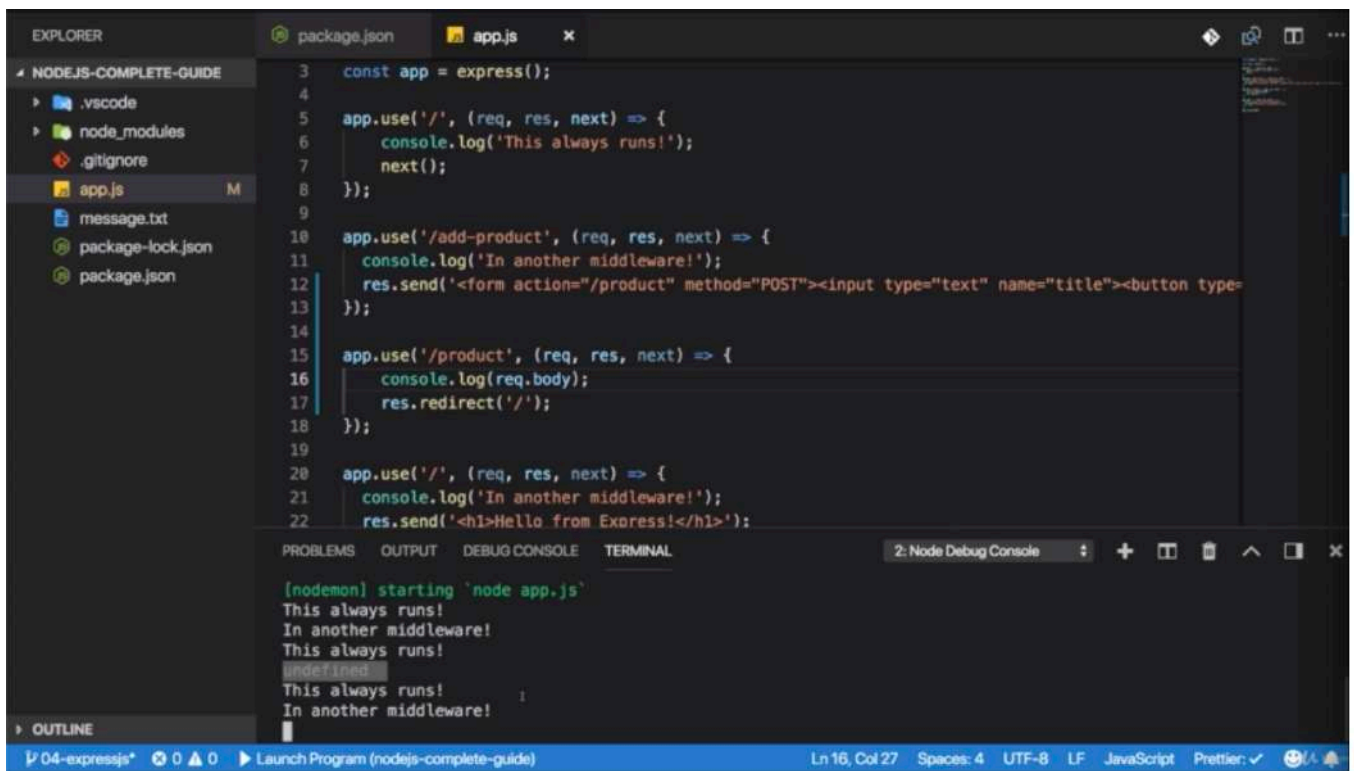
- app.js



Hello from Express!



- if we go back to '/add-product' and fill out input field, we redirect to '/'

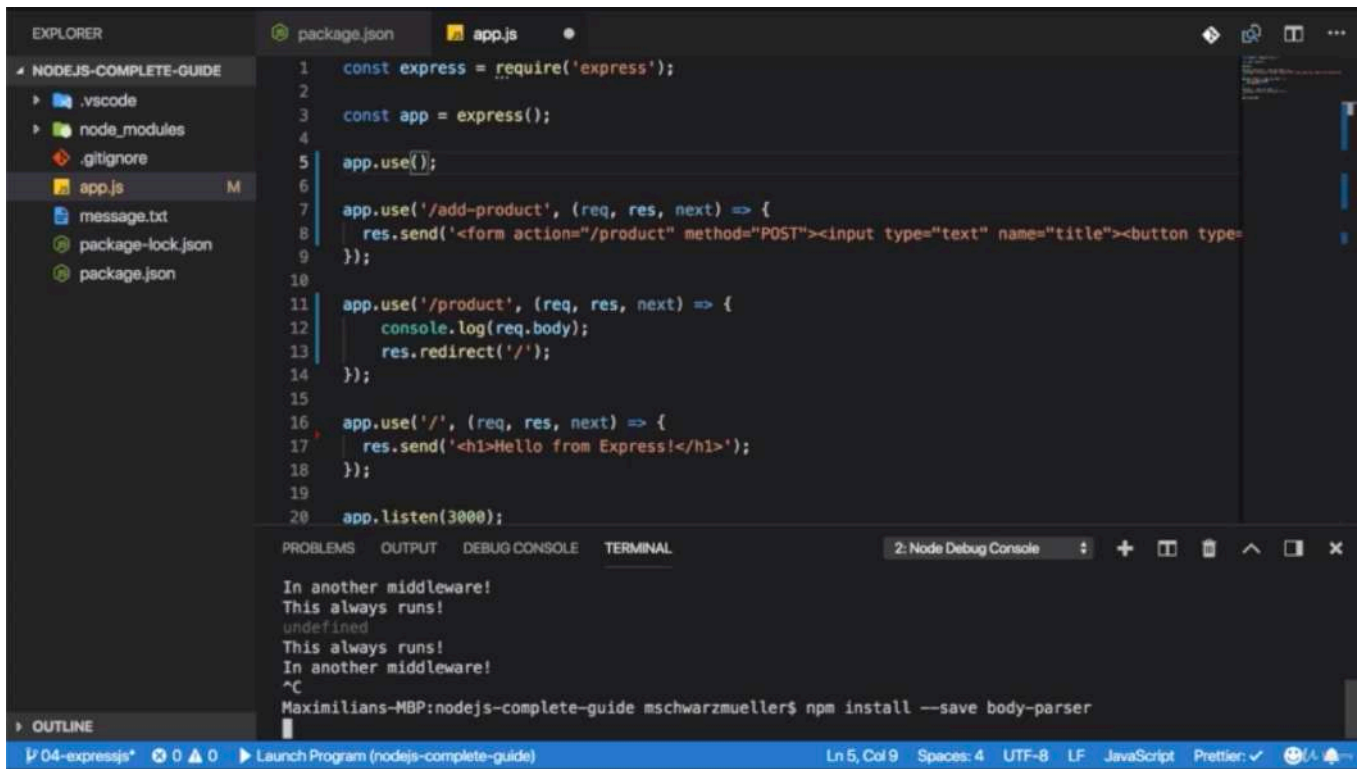


- and we see 'undefined' in the console.
- request gives us .body convenience property. but by default, request doesn't try to parse the incoming request body. so we need to register a parser and we do that by adding another middleware.
- and you typically do that before your route handling middlewares because the parsing of the body should be done no matter where your request ends up. and there i wanna parse the



incoming req.body.

- so we can install a third party package.



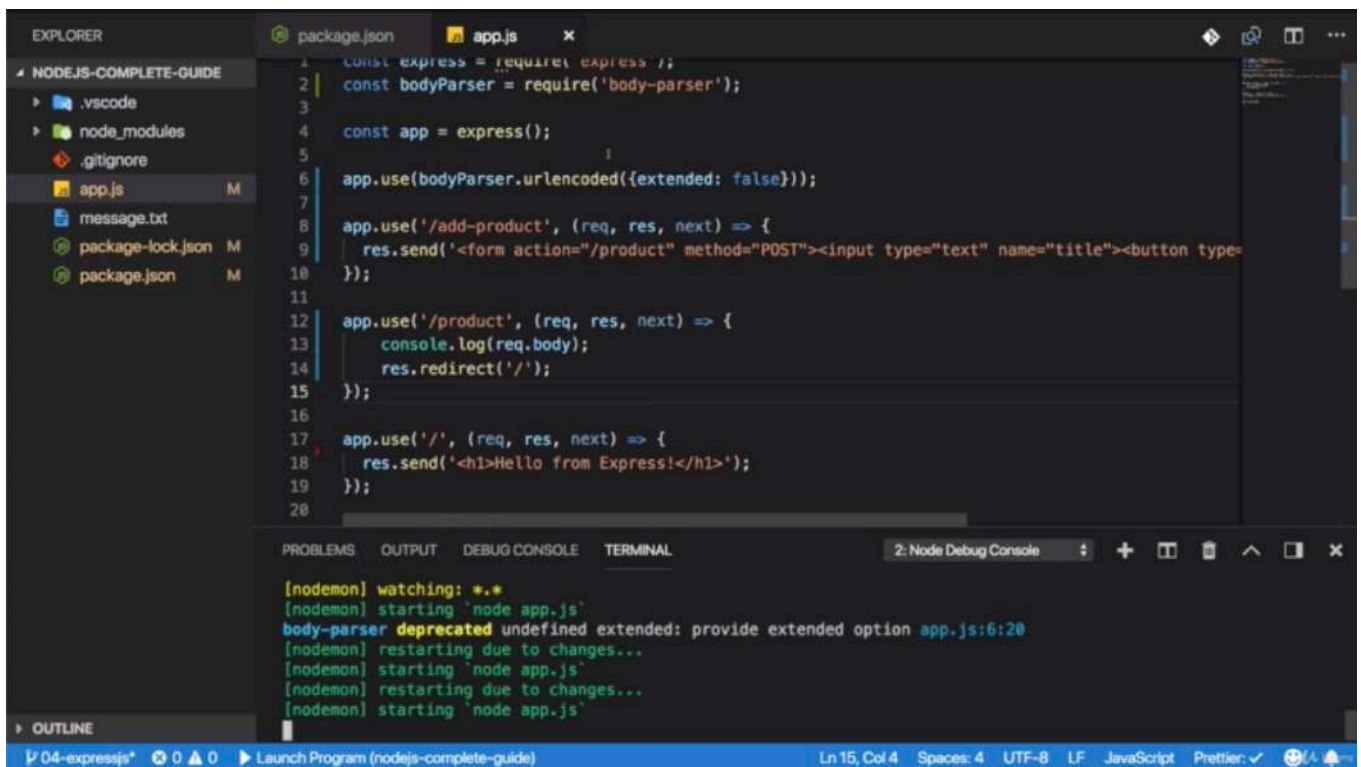
```
1 const express = require('express');
2
3 const app = express();
4
5 app.use();
6
7 app.use('/add-product', (req, res, next) => {
8   res.send('<form action="/product" method="POST"><input type="text" name="title"><button type=
9 });
10
11 app.use('/product', (req, res, next) => {
12   console.log(req.body);
13   res.redirect('/');
14 });
15
16 app.use('/', (req, res, next) => {
17   res.send('<h1>Hello from Express!</h1>');
18 });
19
20 app.listen(3000);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

2: Node Debug Console

In another middleware!  
This always runs!  
undefined  
This always runs!  
In another middleware!  
^C  
Maximilians-MBP:nodejs-complete-guide mschwarzmueller\$ npm install --save body-parser

- —save because this will also be a package that is used in our code that does matter for production. and the package name is 'body-parser'



```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3
4 const app = express();
5
6 app.use(bodyParser.urlencoded({extended: false}));
7
8 app.use('/add-product', (req, res, next) => {
9   res.send('<form action="/product" method="POST"><input type="text" name="title"><button type=
10 });
11
12 app.use('/product', (req, res, next) => {
13   console.log(req.body);
14   res.redirect('/');
15 });
16
17 app.use('/', (req, res, next) => {
18   res.send('<h1>Hello from Express!</h1>');
19 });
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

2: Node Debug Console

[nodemon] watching: \*.\*  
[nodemon] starting 'node app.js'  
body-parser deprecated undefined extended: provide extended option app.js:6:20  
[nodemon] restarting due to changes...  
[nodemon] starting 'node app.js'  
[nodemon] restarting due to changes...  
[nodemon] starting 'node app.js'

- we get that body parser enabled

Book Add Product

Elements Console Network >> View: [Icons] Group by frame [X] Preserve [X]

Filter [X] Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other

Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	...	Size	...	Waterfall	▲
add-product	200	do...	...	325 B	...		
ng-validate.js	200	scr...	...	(from dis...	...		

2 requests | 325 B transferred | Finish: 215 ms | DOMContentLoaded: 120 m...

Console What's New X

Highlights from the Chrome 68 update

- and go back '/add-product' again, fill out input field.

Hello from Express!

Elements Console Network >> View: [Icons] Group by frame [X] Preserve [X]

Filter [X] Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other

Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	...	Size	...	Waterfall	▲
product	302	tex...	...	193 B	...		
localhost	200	do...	...	233 B	...		
ng-validate.js	200	scr...	...	(from dis...	...		

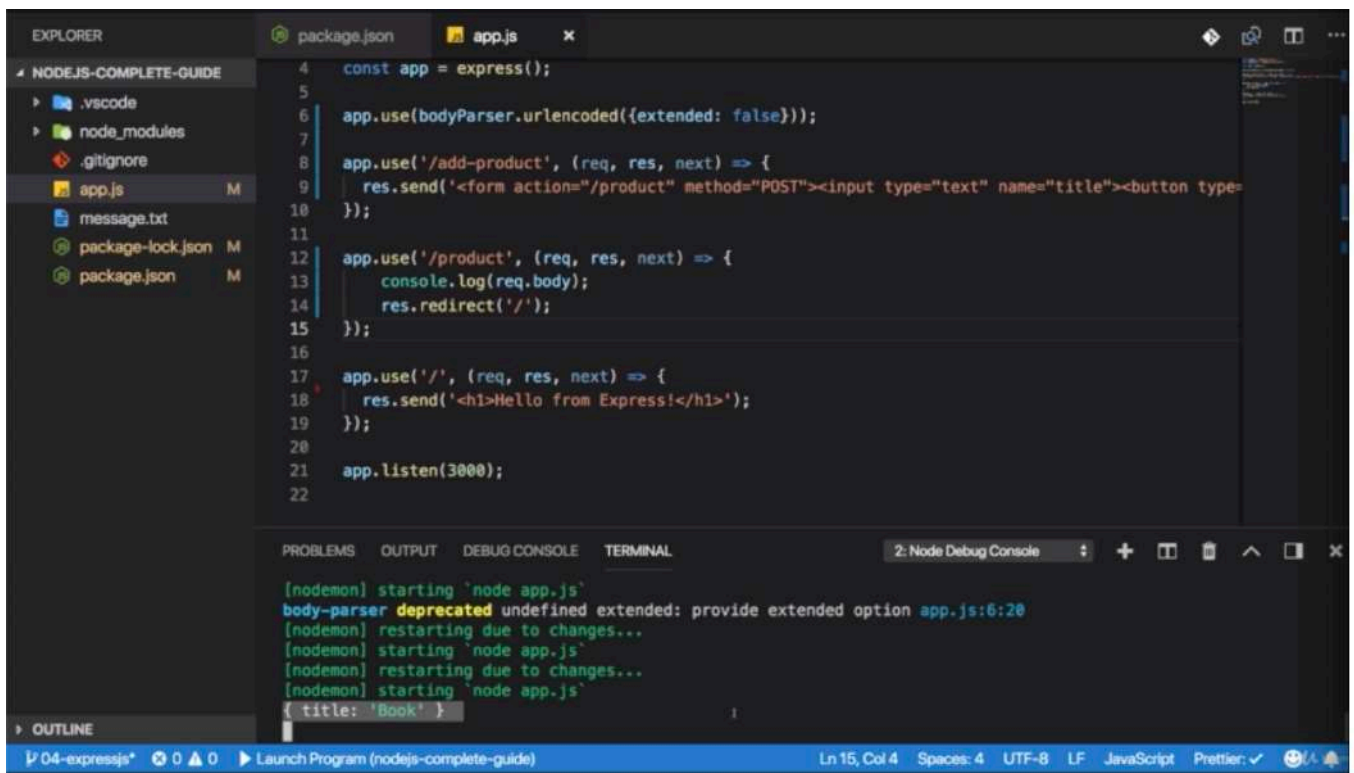
3 requests | 426 B transferred | Finish: 220 ms | DOMContentLoaded: 115 m...

Console What's New X

Highlights from the Chrome 68 update

- and it redirect.



A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project structure with files like .vscode, node\_modules, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor window displays the content of app.js, which is a Node.js Express application. The code includes imports for express and body-parser, middleware setup for urlencoded(), route handlers for /add-product and /product, and a default route for /. The terminal at the bottom shows the output of running the application, including messages from nodemon and the body-parser deprecation warning. The status bar at the bottom indicates the file is at line 15, column 4, using UTF-8 encoding and LF line endings.

- and we see this is what we get, a javascript object with a key-value pair which also makes extracting the value easier than we had to do before with the split function where we manually had to create that array and so on.

```
//app.js

const express = require('express');
const bodyParser = require('body-parser');

const app = express()

/**'urlencoded()' is a function you have to execute
 * and you can pass options to configure it
 * but you don't have to here
 *
 * and 'bodyParser' is the middleware
 * so 'urlencoded()' function in the end just yields us such a middleware function
 * so 'bodyParser.urlencoded()' parse such a function like (req, res, next) => {}
 * and in the end, this middleware function call 'next' in the end,
 * so that the request also reaches our middleware
 *
 * but before it does that, it will do that whole request body parsing we had to do
 *
 * now this will not parse all kinds of possible bodies, files, json and so on
 * but this will parse bodies like the one which is sent through a form.
 */

/** in 'urlencoded()', you should pass the config options
 * '{extended: false}' is added to comply with what we should use here
 */
app.use(bodyParser.urlencoded({extended: false}))
```

```

app.use('/add-product', (req, res, next) => {
  /**the path, the URL to which the request should be sent */
  res.send('<form action="/product" method="POST"><input type="text" name="title'
})

/**'/product' has to be after '/add-product' because of preventing '/product' from
 * '/' has to be after '/product' because of preventing '/' from overlapping to '/'
 */
app.use('/product', (req, res, next) => {
  /**'req.body' is a new field added by express */
  console.log(req.body)
  /**i can use res.redirect which certainly is easier than manually setting the
   *
   */
  res.redirect('/');
})

app.use('/', (req, res, next) => {
  res.send('<h1>Hello from Express!</h1>')
})

app.listen(3000);

```

## 🔗 \* Chapter 64: Limiting Middleware Execution To POST Requests

---

### 1. update

- app.js

The screenshot shows the VS Code editor with the following code in `app.js`:

```
4 const app = express();
5
6 app.use(bodyParser.urlencoded({extended: false}));
7
8 app.use('/add-product', (req, res, next) => {
9   res.send('<form action="/product" method="POST"><input type="text" name="title"><button type=
10 });
11
12 app.post('/product', (req, res, next) => {
13   console.log(req.body);
14   res.redirect('/');
15 });
16
17 app.use('/', (req, res, next) => {
18   res.send('<h1>Hello from Express!</h1>');
19 });
20
21 app.listen(3000);
22
```

The terminal output shows the following logs:

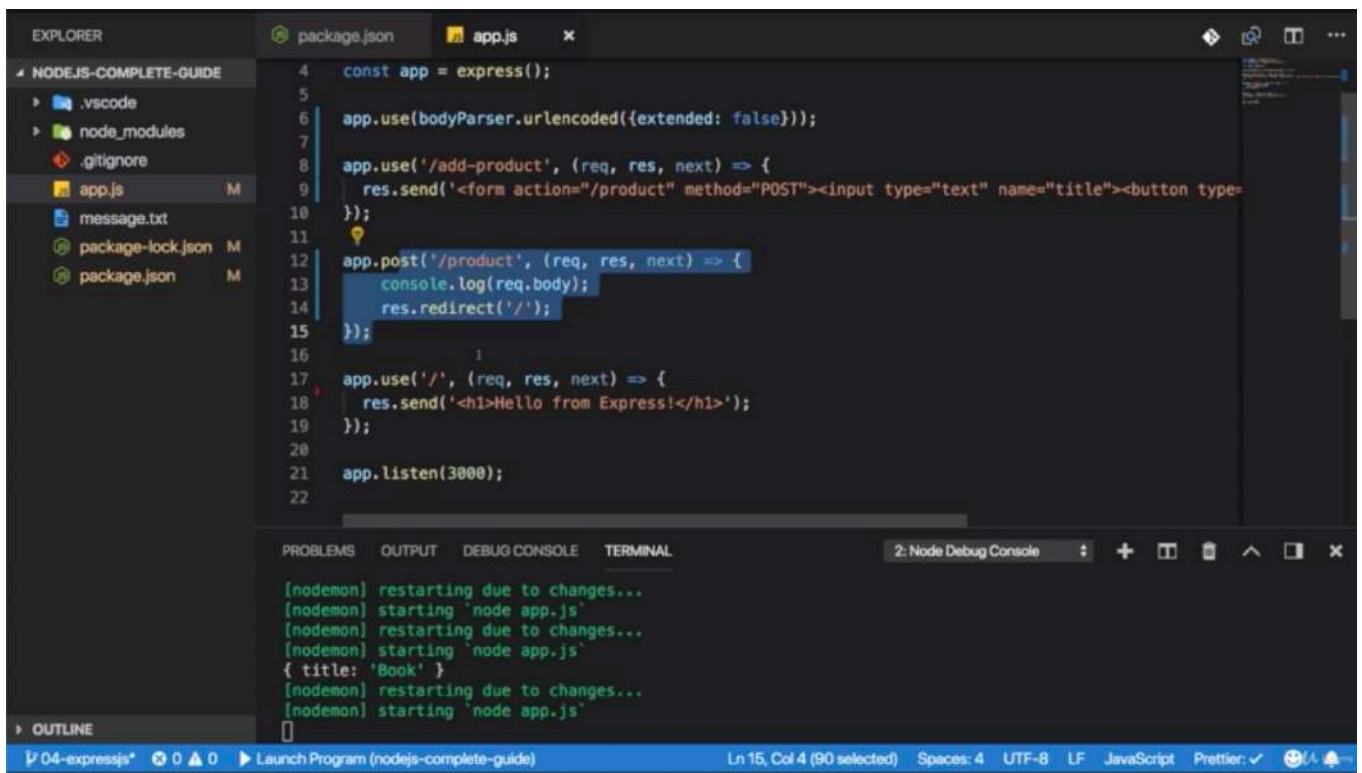
```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
{ title: 'Book' }
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

Hello from Express!

The screenshot shows the Chrome DevTools Network tab with the following data:

Name	Status	Type	Size	Waterfall
product	200	do...	233 B	
ng-validate.js	200	scr...	(from dis...)	

2 requests | 233 B transferred | Finish: 237 ms | DOMContentLoaded: 133 m...



- if i go to '/product', i see 'hello from express' so i don't end up here

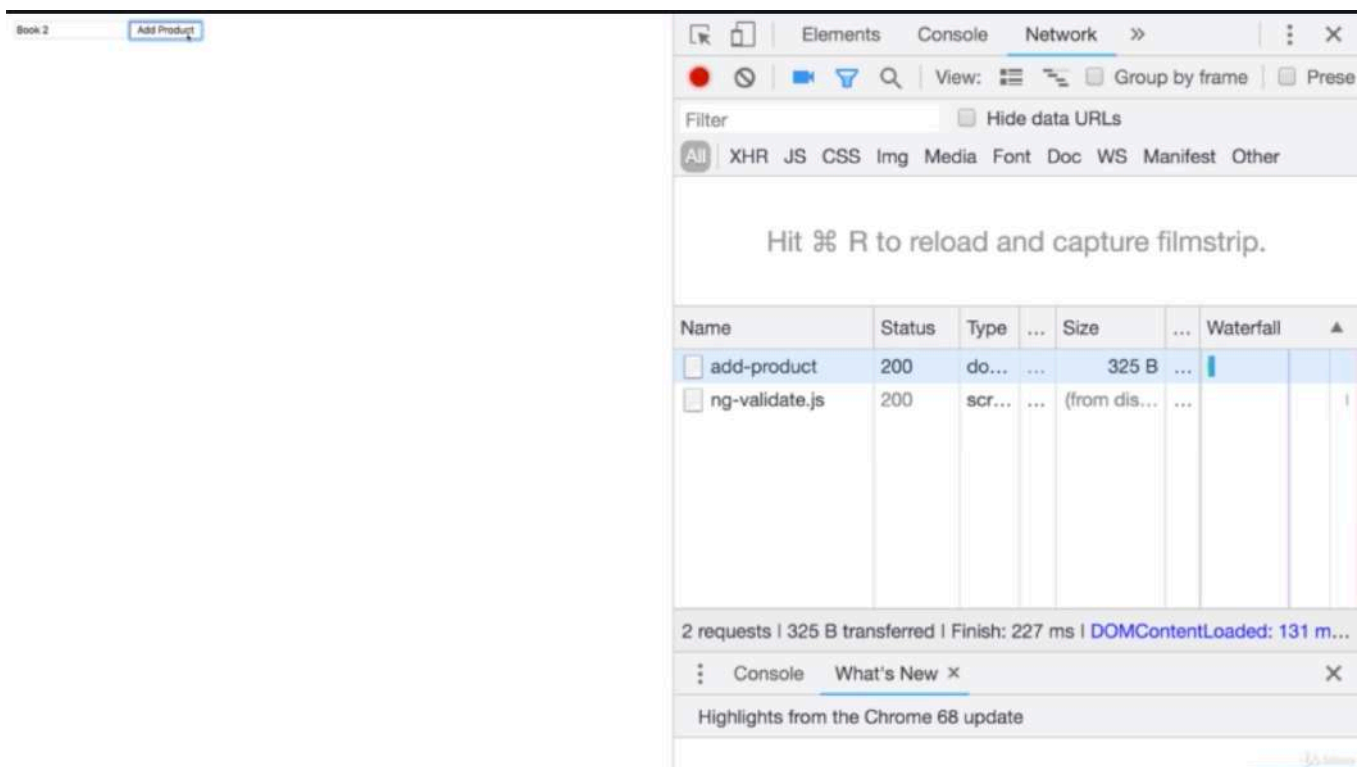
//app.js

```

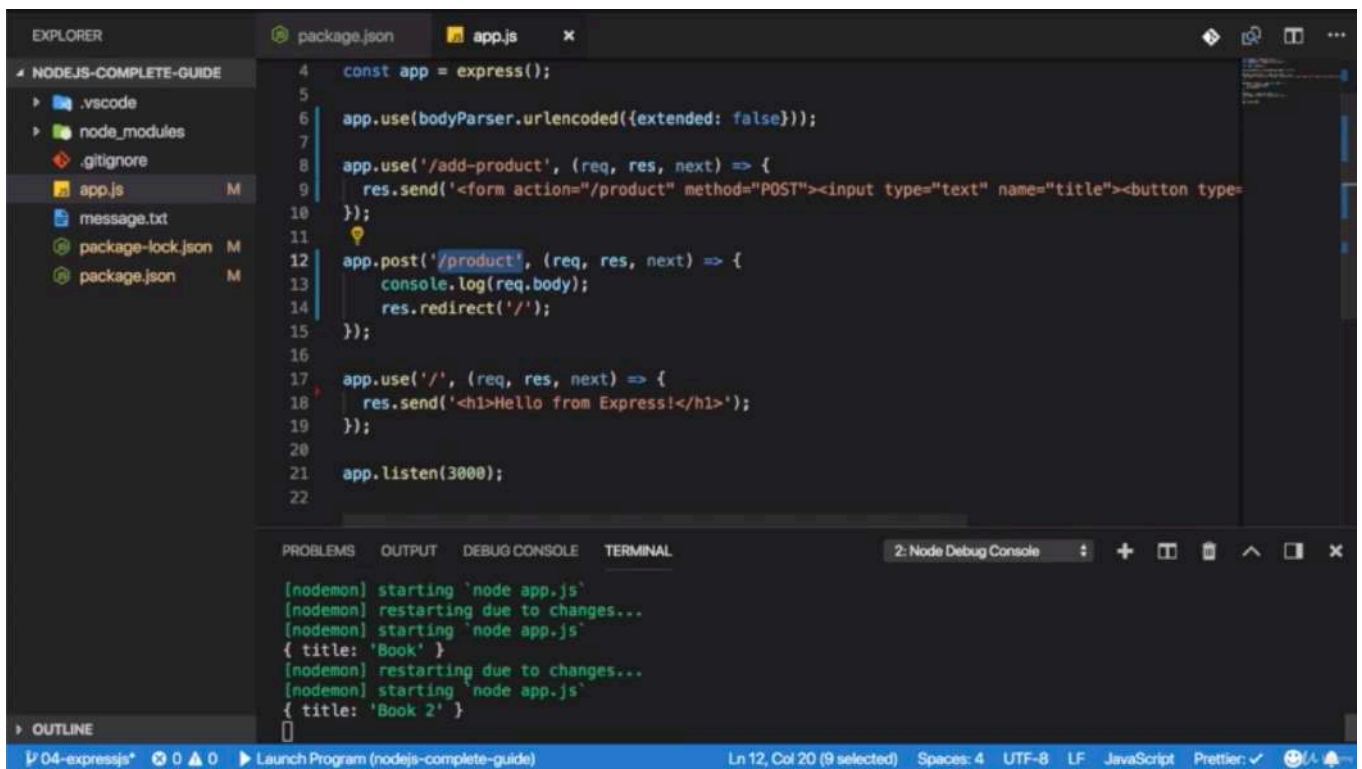
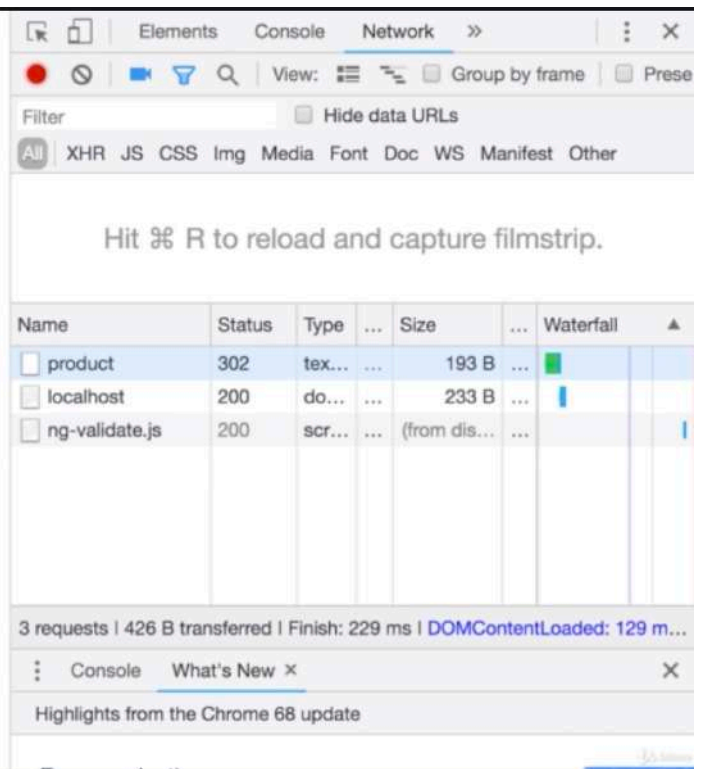
app.post('/product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/');
})

```

even though i entered '/product' but it was a GET request.



Hello from Express!



- but if i send POST request through that form, '/add-product', you see we get this output 'book 2'. so we clearly made it into this below middleware due to our filtering.

```
//app.js

app.post('/product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/')
})
```

- so this is another way of using that middleware function, instead of 'use()' which will work with all http methods.
- we can also use get or post to filter for these.

```
//app.js

const express = require('express');
const bodyParser = require('body-parser');

const app = express()

app.use(bodyParser.urlencoded({extended: false}))

app.use('/add-product', (req, res, next) => {
  res.send('<form action="/product" method="POST"><input type="text" name="title">')
})

/**this middleware always execute, not just for POST requests but also for GET requests
 * what can we do regarding that?
 * we can use app.get() which is basically app.use(), it has the same syntax as app.use()
 * it only will fire for incoming GET request.
 * this is another form of filtering besides filtering for the path, app.get allows us to filter by method
 */

/**'app.post()' to filter for incoming POST requests
 * and this 'app.post()' will only trigger for incoming POST requests with this path
 * and not for GET request. */
app.post('/product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/');
})

app.use('/', (req, res, next) => {
  res.send('<h1>Hello from Express!</h1>')
})

app.listen(3000);
```

## \* Chapter 65: Using Express Router

### 1. update

- app.js
- ./routes/admin.js
- ./routes/shop.js



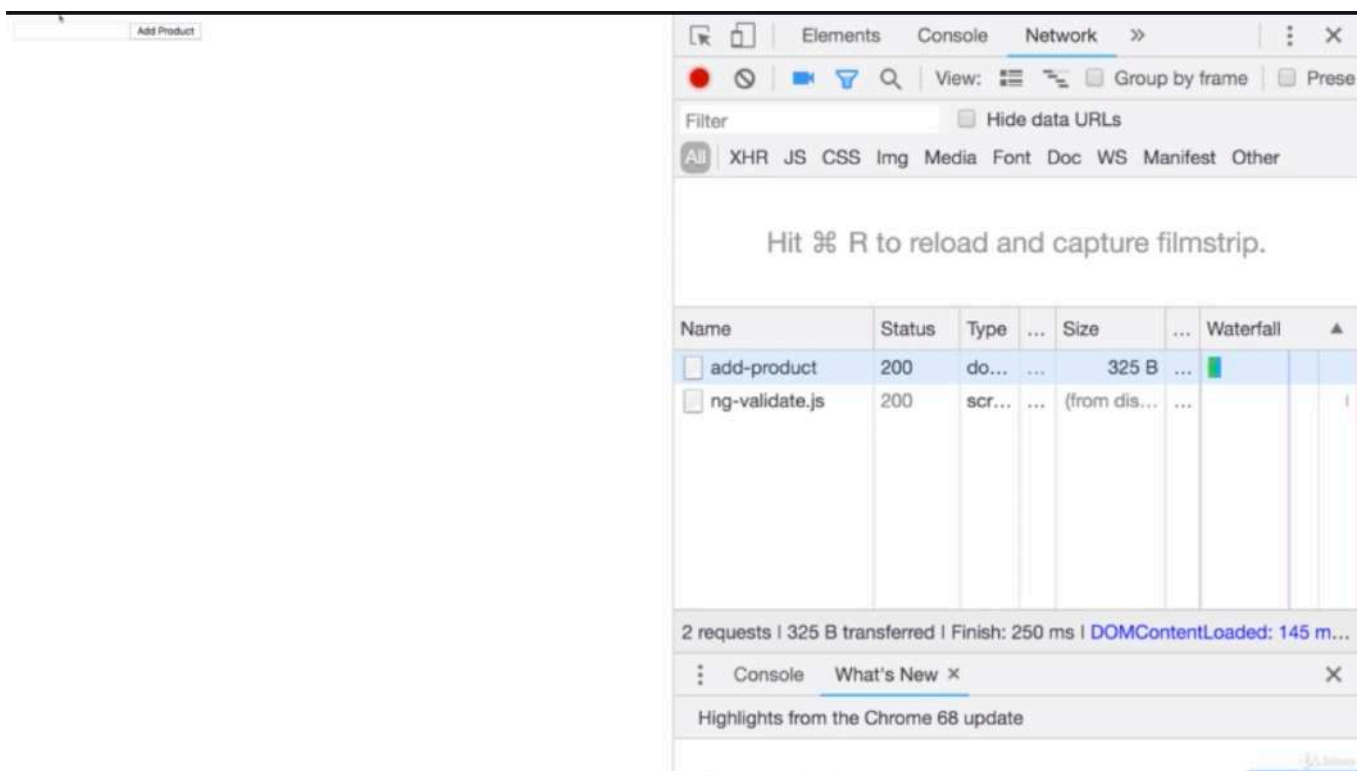
The screenshot shows a VS Code editor with a file explorer on the left containing files like .vscode, node\_modules, routes, admin.js, shop.js, .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor displays app.js with the following code:

```
4 const app = express();
5
6 const adminRoutes = require('./routes/admin');
7
8 app.use(bodyParser.urlencoded({extended: false}));
9
10
11
12 app.use('/', (req, res, next) => {
13   res.send('<h1>Hello from Express!</h1>');
14 });
15
16 app.use(adminRoutes);
17
18 app.listen(3000);
19
```

The terminal at the bottom shows the following output:

```
{ title: 'Book 2' }
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- the order matters. so if we put this after this middleware, we will never reach that.



- if i save and reload, this works.

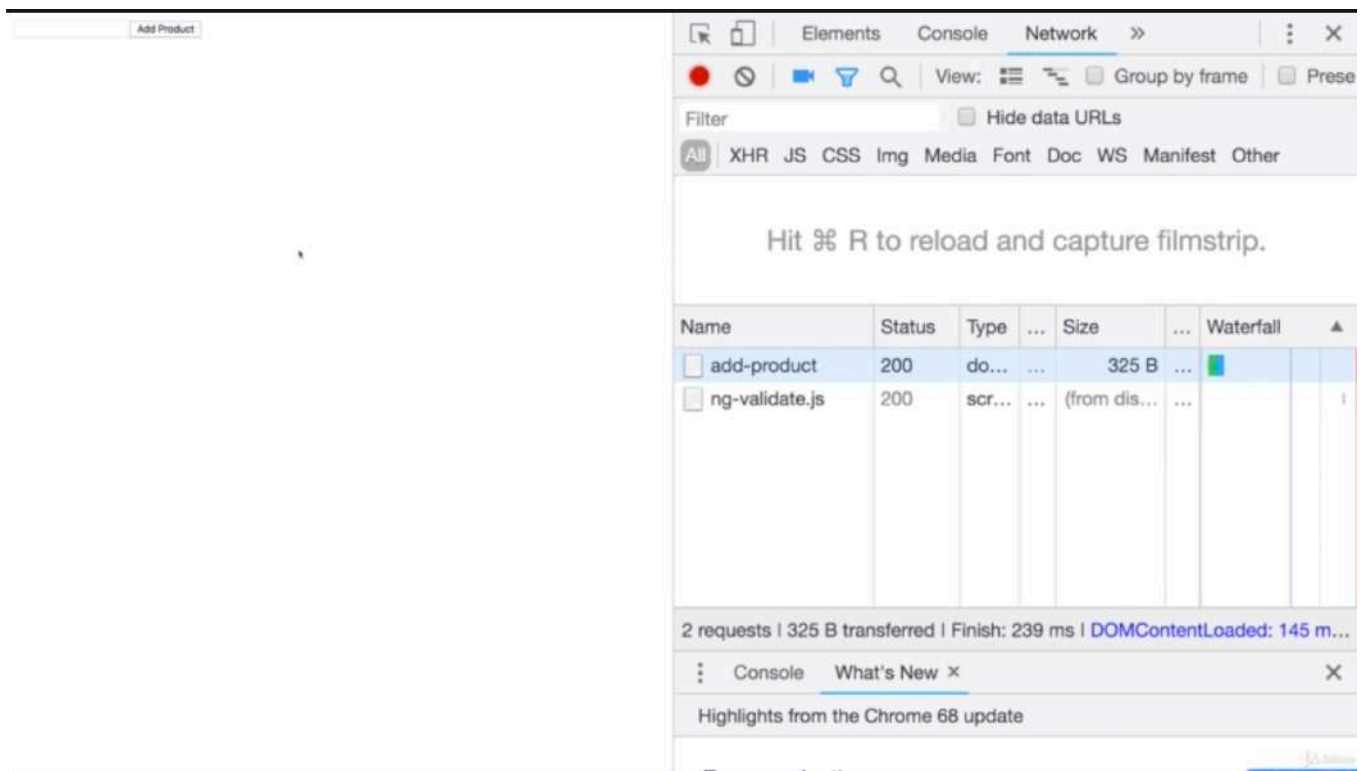
The screenshot shows the VS Code editor with the `app.js` file open. The file contains the following code:

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3
4 const app = express();
5
6 const adminRoutes = require('./routes/admin');
7 const shopRoutes = require('./routes/shop');
8
9 app.use(bodyParser.urlencoded({extended: false}));
10
11 app.use(shopRoutes);
12 app.use(adminRoutes);
13
14 app.listen(3000);
15
```

The terminal output shows the following messages:

```
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- if i would switch the position of `'app.use(shopRoutes)'` and `'app.use(adminRoutes)'`





```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.get('/', (req, res, next) => {
6   res.send('<h1>Hello from Express!</h1>');
7 });
8
9 module.exports = router;
```

Terminal output:

```
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- and i reload, it would work and we would not end up in this route
- this only happens because i have .get() here. get, post and so on will actually do an exact match here.

```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.use('/', (req, res, next) => {
6   res.send('use');
7 });
8
9 module.exports = router;
```

Tooltip text:

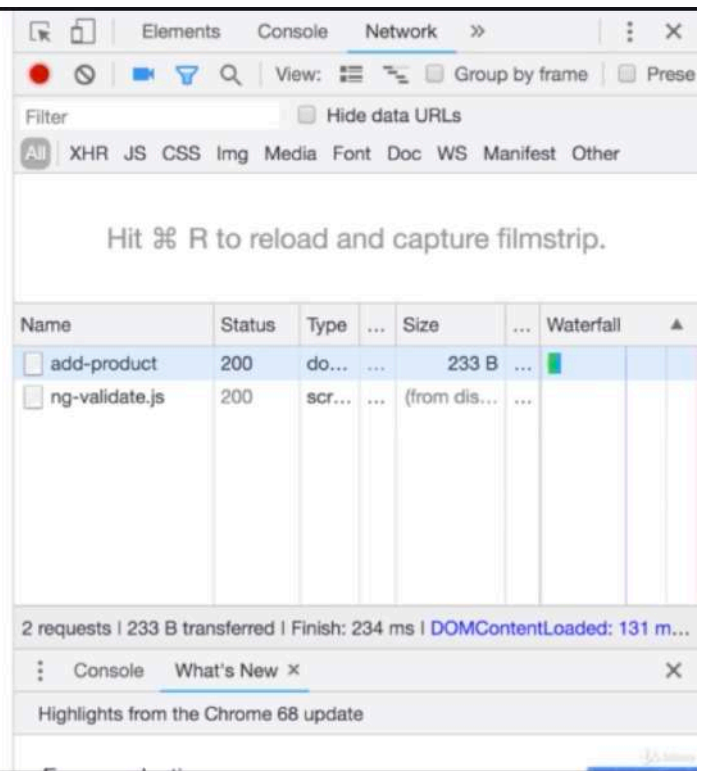
```
(property) IRouter.use: (...handlers: Req...estHandler[]) => Router (+3 overloads)
```

Terminal output:

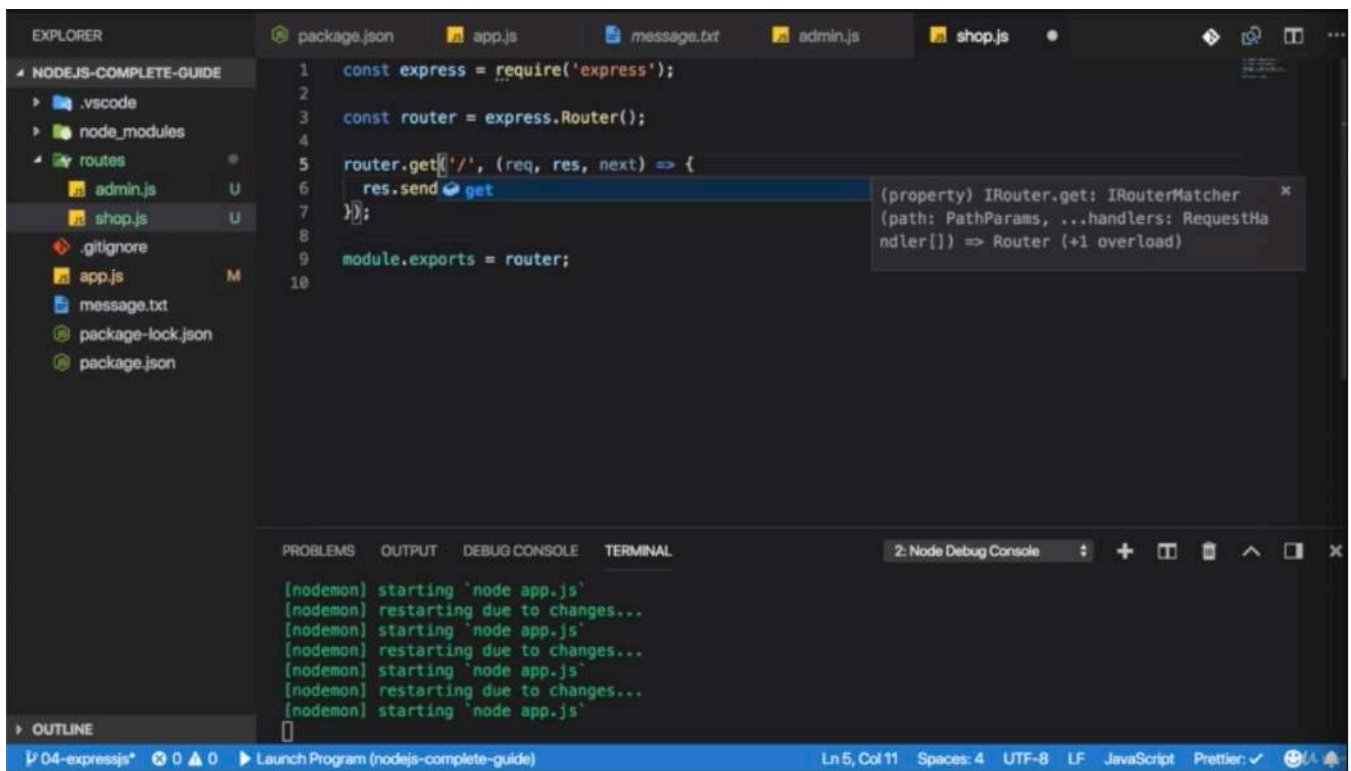
```
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- if i would use 'use()' here as i did before to handle any incoming http method,

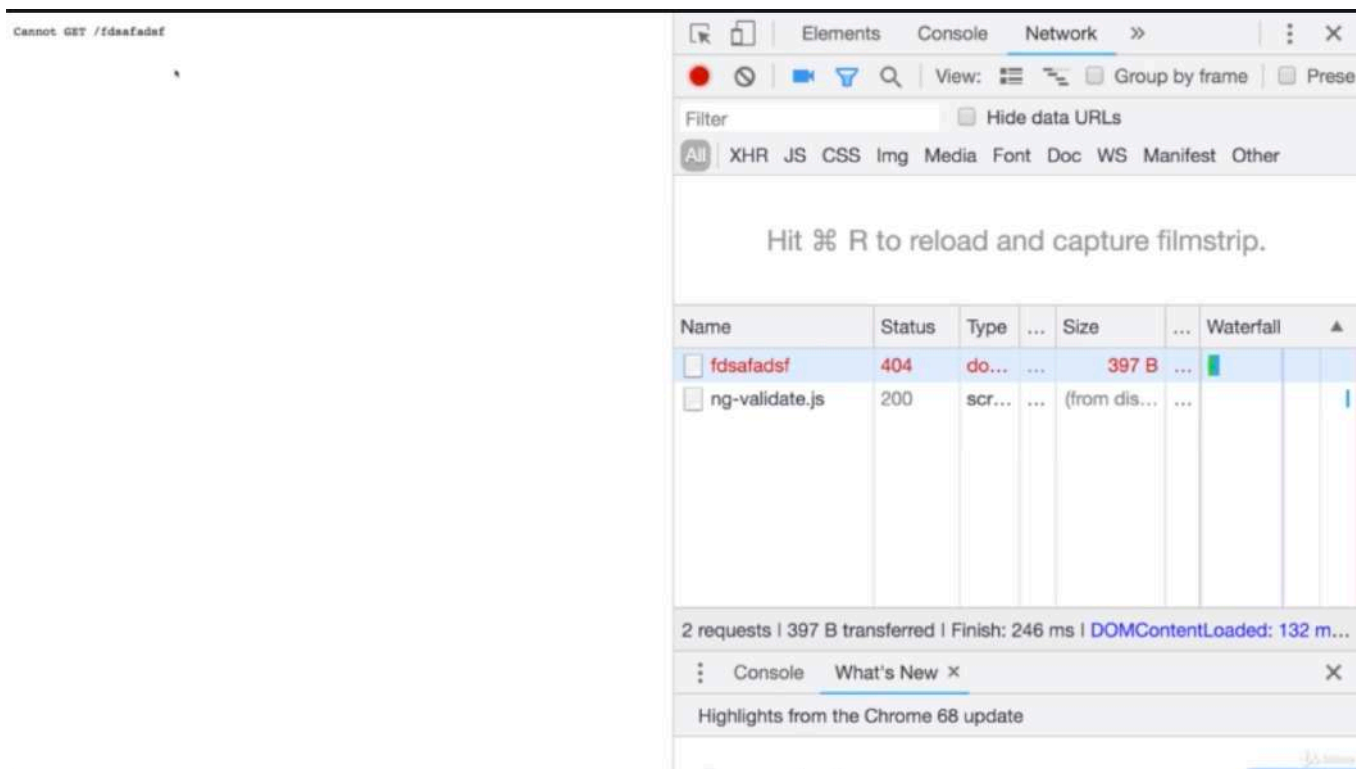
Hello from Express!



- then if i reload here, we see hello from express again.



- so this exact matching is not achieved by using the router but because we use get here, and that would have been the same if we stick to app way of doing this in the app.js file we had previously
- so 'get()' method make sure that it's not just a get method but this exact path



- and therefore now if i enter some random stuff, i get an error because now i got no single middleware that would handle that stuff.

```
//app.js

/**the order of import doesn't matter */
const express = require('express');
const bodyParser = require('body-parser');

const app = express()

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}))

/**this order matters */
app.use(adminRoutes);
app.use(shopRoutes);

app.listen(3000);
```

```
//./routes/admin.js

/**this should be the route that handles the creation of products which the admin

const express = require('express')

/** 'Router()' is like a mini express app tied to the other express app or plugga
const router = express.Router();
```

```

/**'router' can be used to again define a use() function for all requests,
 * get function for GET, post function for POST
 *
 * 'router' functions basically work in exactly the same way as the app.use function
 * or the app.get() and so on.
 *
 * i will rename this to 'get()' because i only wanna handle GET requests to 'add-product'
 * and return this form.
 */
router.get('/add-product', (req, res, next) => {
  res.send('<form action="/product" method="POST"><input type="text" name="title">')
})

router.post('/product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/');
})

/**with that, we can import that into the app.js file
 *
 * 'router()' is a valid middleware function.
 * so we can take admin routes and just call app.use and put our admin routes in there
 */
module.exports = router;

```

```

//./routes/shop.js

const express = require('express');

const router = express.Router();

router.get('/', (req, res, next) => {
  res.send('<h1>Hello from Express!</h1>')
})

module.exports = router

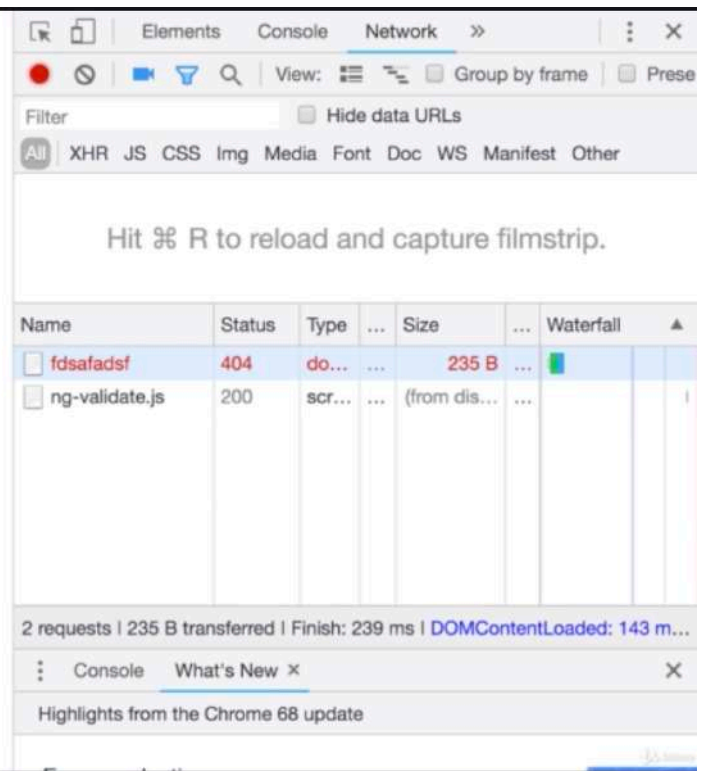
```

## \* Chapter 66: Adding A 404 Error Page

### 1. update

- app.js

Page not found



```
//app.js

/**the order of import doesn't matter */
const express = require('express');
const bodyParser = require('body-parser');

const app = express()

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}))

app.use(adminRoutes);
app.use(shopRoutes);

/**if we got no fitting middleware and we don't have one here,
 * then we make it all the way to the bottom
 * and eventually we don't handle that request.
 * so to send 404 error page, we simply have to add a catch all middleware at the
 */

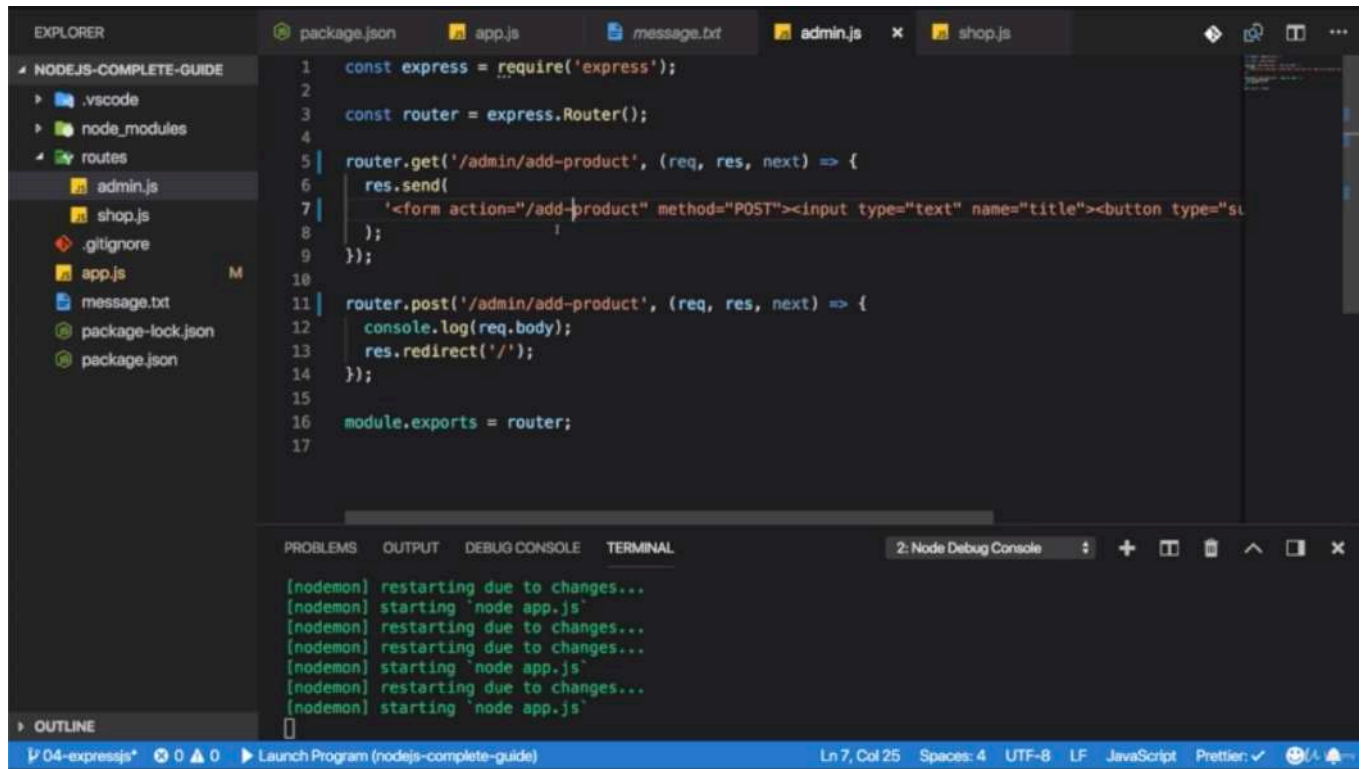
/**maybe we also wanna set the 404 status code
 * and you can do that by chaining another method prior to send
 * and that is the status() method.
 */
app.use((req, res, next) => {
  res.status(404).send('<h1>Page not found</h1>')
})

app.listen(3000);
```

## \* Chapter 67: Filtering Paths

### 1. update

- app.js
- ./routes/admin.js
- ./routes/shop.js



The screenshot shows a VS Code editor with the following files open: package.json, app.js, message.txt, admin.js, and shop.js. The Explorer sidebar on the left shows the project structure: NODEJS-COMPLETE-GUIDE, .vscode, node\_modules, routes (containing admin.js and shop.js), .gitignore, app.js, message.txt, package-lock.json, and package.json. The main editor displays the code in app.js, which uses Express.js to set up a router. The code defines a GET route for '/admin/add-product' that sends an HTML form, and a POST route for the same path that logs the request body and redirects to the root. The terminal at the bottom shows the output of the Node.js application, indicating that it is restarting due to changes and starting successfully.

```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.get('/admin/add-product', (req, res, next) => {
6   res.send(
7     '<form action="/admin/add-product" method="POST"><input type="text" name="title"><button type="su
8   );
9 });
10
11 router.post('/admin/add-product', (req, res, next) => {
12   console.log(req.body);
13   res.redirect('/');
14 });
15
16 module.exports = router;
17
```

2: Node Debug Console

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- the same path can be used if the methods differ.



The screenshot shows a VS Code editor with a project named '04-expressjs'. The Explorer sidebar on the left shows a file structure with 'routes' containing 'admin.js' and 'shop.js'. The main editor displays 'app.js' with the following code:

```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.get('/admin/add-product', (req, res, next) => {
6   res.send(
7     '<form action="/add-product" method="POST"><input type="text" name="title"><button type="su
8   );
9 });
10
11 router.post('/admin/add-product', (req, res, next) => {
12   console.log(req.body);
13   res.redirect('/');
14 });
15
16 module.exports = router;
```

The bottom panel shows the '2: Node Debug Console' with the following output:

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- if we have such a setup where our paths in such a router file start with the same part or with the same segment '/admin', we can take that segment out of this route

The screenshot shows the same VS Code editor with 'app.js' updated to use route filtering. The code is as follows:

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3
4 const app = express();
5
6 const adminRoutes = require('./routes/admin');
7 const shopRoutes = require('./routes/shop');
8
9 app.use(bodyParser.urlencoded({extended: false}));
10
11 app.use('/admin', adminRoutes);
12 app.use(shopRoutes);
13
14 app.use((req, res, next) => {
15   res.status(404).send('<h1>Page not found</h1>');
16 });
17
18 app.listen(3000);
```

The bottom panel shows the '2: Node Debug Console' with the following output:

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- and then go to the app.js file and add it here, so add that segment as a filter.
- only routes starting with '/admin' will go into the admin routes file.

```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.get('/add-product', (req, res, next) => {
6   res.send(
7     '<form action="/add-product" method="POST"><input type="text" name="title"><button type="submit">Add Product</button></form>';
8   );
9 });
10
11 router.post('/add-product', (req, res, next) => {
12   console.log(req.body);
13   res.redirect('/');
14 });
15
16 module.exports = router;
17
```

2: Node Debug Console

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- so now '/add-product' will match the '/admin/add-product' route because '/admin' was already stripped out.

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3
4 const app = express();
5
6 const adminRoutes = require('./routes/admin');
7 const shopRoutes = require('./routes/shop');
8
9 app.use(bodyParser.urlencoded({extended: false}));
10
11 app.use('/admin', adminRoutes);
12 app.use(shopRoutes);
13
14 app.use((req, res, next) => {
15   res.status(404).send('<h1>Page not found</h1>');
16 });
17
18 app.listen(3000);
19
```

2: Node Debug Console

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```



## Page not found

The screenshot shows the Chrome DevTools Network tab. The top toolbar includes icons for Elements, Console, Network, and a search icon. The 'Network' tab is selected, and the 'Filter' dropdown is set to 'All'. The 'Hide data URLs' checkbox is checked. The 'View' dropdown is set to 'Pre-rendered'. The 'Group by frame' checkbox is checked. The 'Preserve log' checkbox is checked. The 'Filter' input field is empty. The 'All' filter is selected. The 'XHR' filter is selected. The 'JS' filter is selected. The 'CSS' filter is selected. The 'Img' filter is selected. The 'Media' filter is selected. The 'Font' filter is selected. The 'Doc' filter is selected. The 'WS' filter is selected. The 'Manifest' filter is selected. The 'Other' filter is selected. The '9 ms' and '221 ms' labels are visible. The 'Name' column shows 'add-product' and 'ng-validate.js'. The 'Status' column shows '404' and '200'. The 'Type' column shows 'do...' and 'scr...'. The 'Size' column shows '235 B' and '(from dis...)'. The 'Waterfall' column shows a green bar for 'add-product' and a blue bar for 'ng-validate.js'. The summary bar shows '2 requests | 235 B transferred | Finish: 243 ms | DOMContentLoaded: 143 m...'. The 'Console' tab is selected, and the 'What's New' message is visible.

Name	Status	Type	...	Size	...	Waterfall
add-product	404	do...	...	235 B	...	
ng-validate.js	200	scr...	...	(from dis...)	...	

2 requests | 235 B transferred | Finish: 243 ms | DOMContentLoaded: 143 m...

Console What's New X

Highlights from the Chrome 68 update

- if i go to just '/add-product', we see 'Page not found' because it doesn't exist anymore.

The screenshot shows the Chrome DevTools Network tab. The top toolbar includes icons for Elements, Console, Network, and a search icon. The 'Network' tab is selected, and the 'Filter' dropdown is set to 'All'. The 'Hide data URLs' checkbox is checked. The 'View' dropdown is set to 'Pre-rendered'. The 'Group by frame' checkbox is checked. The 'Preserve log' checkbox is checked. The 'Filter' input field is empty. The 'All' filter is selected. The 'XHR' filter is selected. The 'JS' filter is selected. The 'CSS' filter is selected. The 'Img' filter is selected. The 'Media' filter is selected. The 'Font' filter is selected. The 'Doc' filter is selected. The 'WS' filter is selected. The 'Manifest' filter is selected. The 'Other' filter is selected. The 'Hit ⌘ R to reload and capture filmstrip.' message is visible. The 'Name' column shows 'add-product' and 'ng-validate.js'. The 'Status' column shows '200' and '200'. The 'Type' column shows 'do...' and 'scr...'. The 'Size' column shows '329 B' and '(from dis...)'. The 'Waterfall' column shows a blue bar for 'add-product' and a blue bar for 'ng-validate.js'. The summary bar shows '2 requests | 329 B transferred | Finish: 223 ms | DOMContentLoaded: 128 m...'. The 'Console' tab is selected, and the 'What's New' message is visible.

Hit ⌘ R to reload and capture filmstrip.

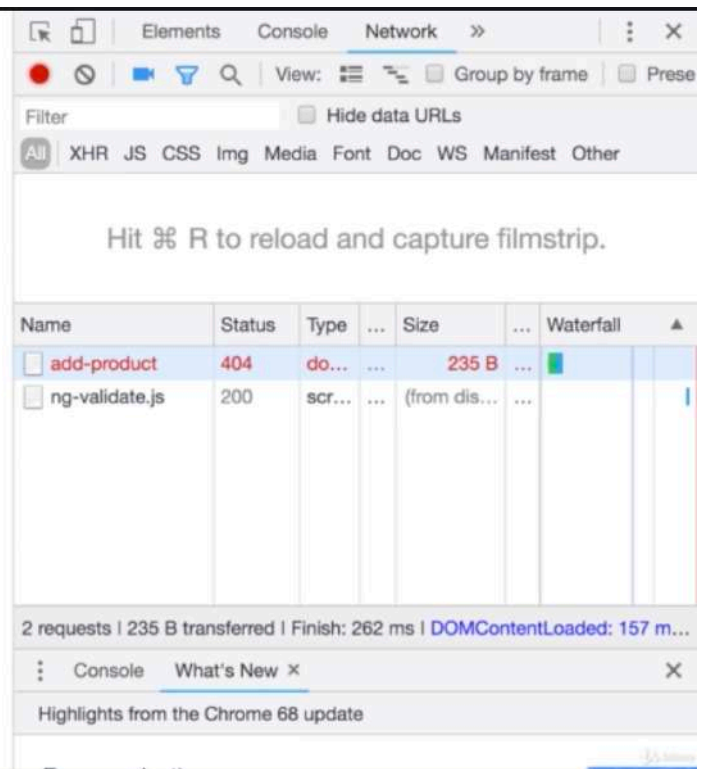
Name	Status	Type	...	Size	...	Waterfall
add-product	200	do...	...	329 B	...	
ng-validate.js	200	scr...	...	(from dis...)	...	

2 requests | 329 B transferred | Finish: 223 ms | DOMContentLoaded: 128 m...

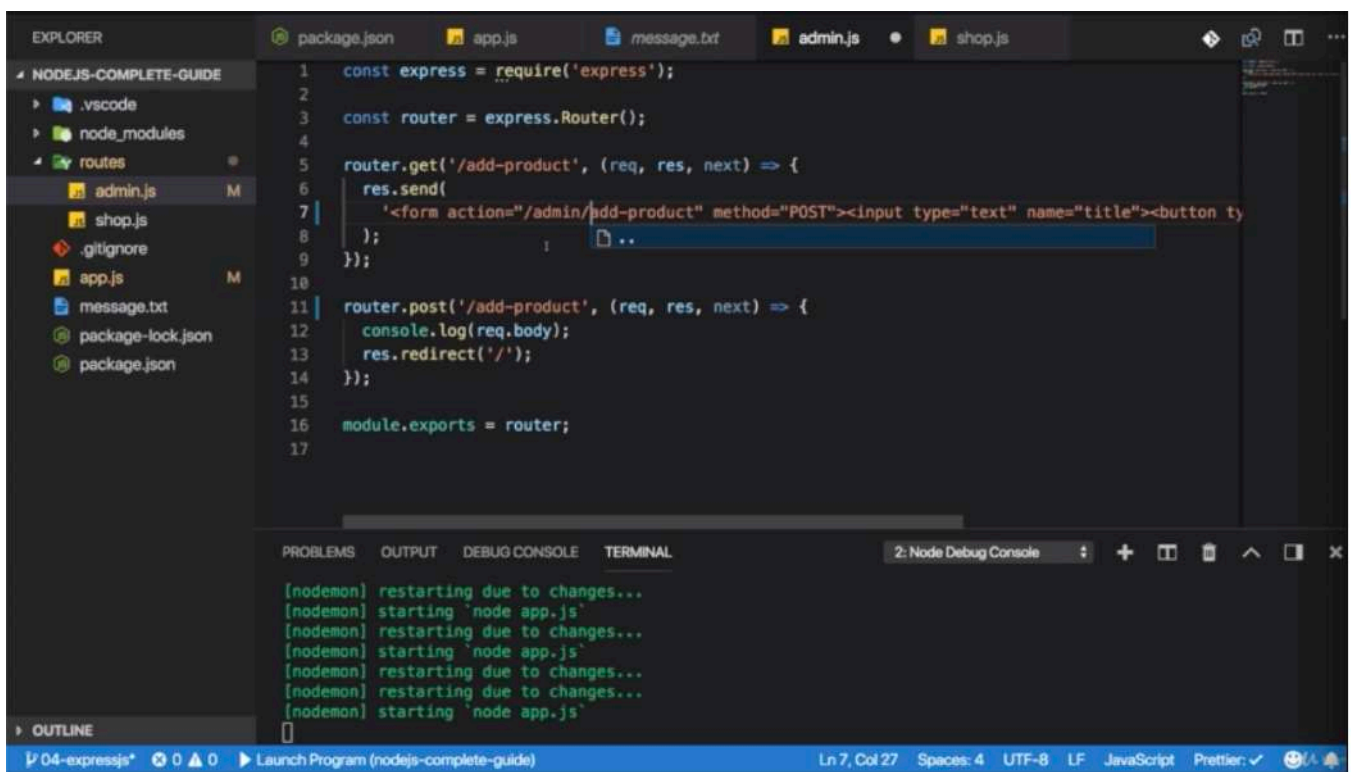
Console What's New X

Highlights from the Chrome 68 update

Page not found



- but if we go to 'admin/add-product', here is the form.
- if i fill out input, i get 'page not found'

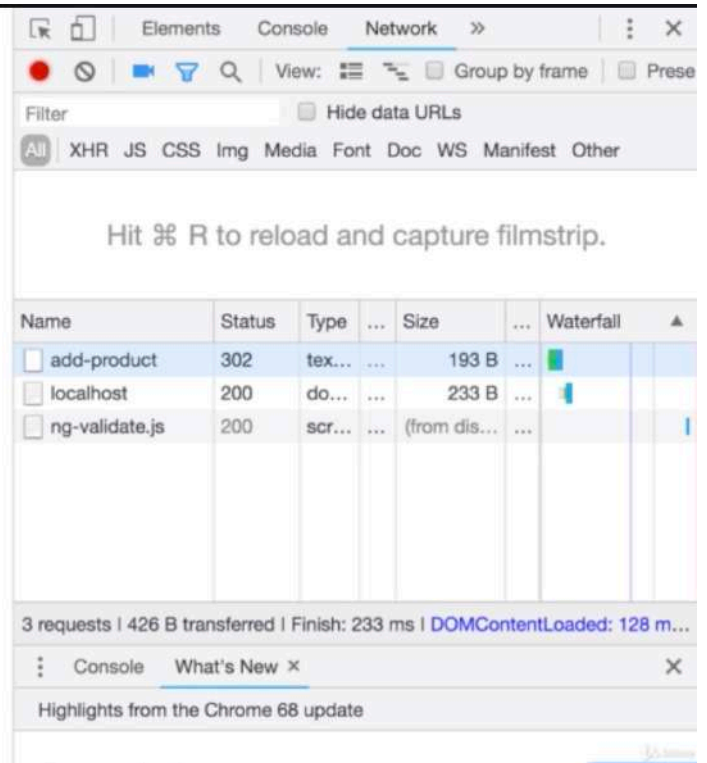


- so this should be done like below. need to be added '/admin' in 'send()' because we wanna reach that route which is the admin.js file which is only reachable through requests that have /admin at the beginning.

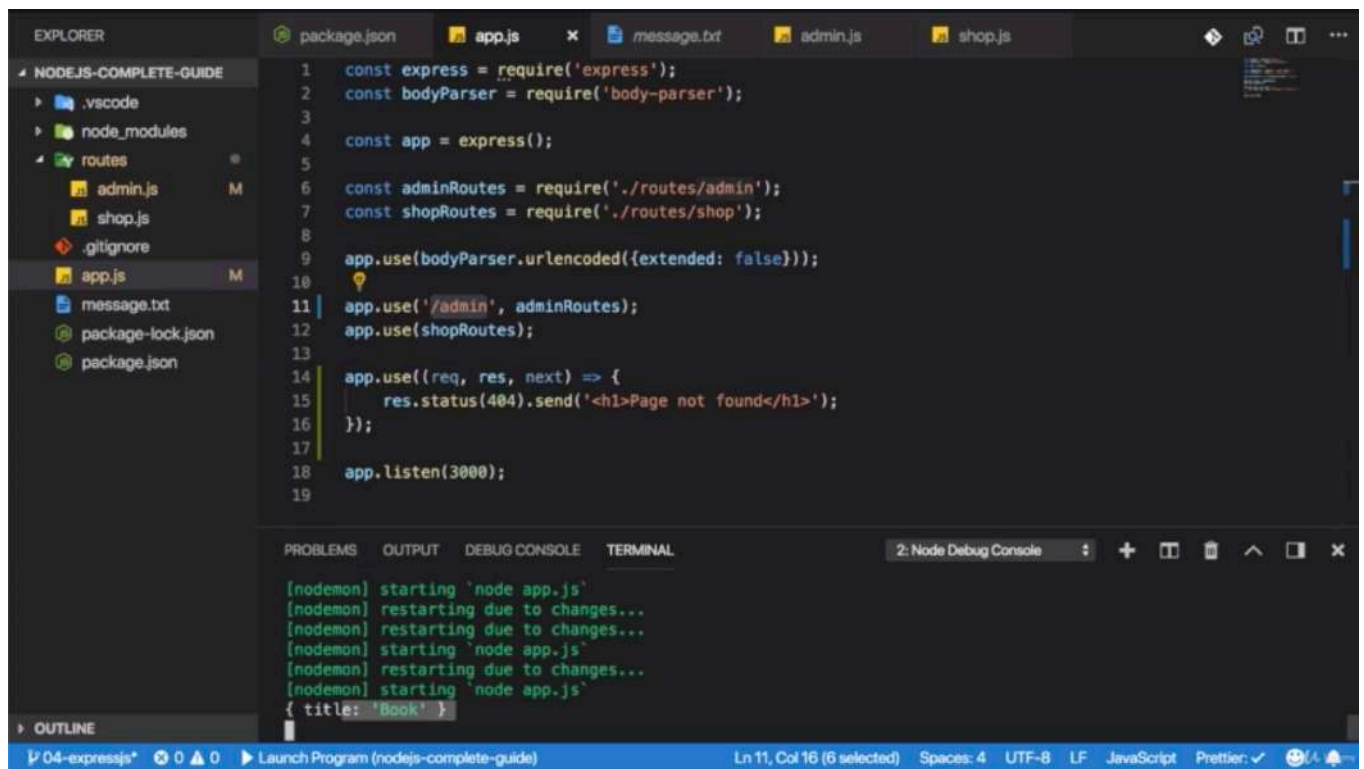
```
///  
./routes/admin.js
```

```
router.get('/add-product', (req, res, next) => {
  res.send('<form action="/admin/add-product" method="POST"><input type="text"
}')
```

Hello from Express!



- so go to '/admin/add-product' and fill out input field, the it's redirected.



- and also see that we are logging this here.

The screenshot shows a VS Code editor with a project named '04-expressjs'. The Explorer sidebar on the left shows the file structure: `node_modules`, `routes` (containing `admin.js` and `shop.js`), `message.txt`, `package-lock.json`, and `package.json`. The main editor displays `app.js` with the following code:

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3
4 const app = express();
5
6 const adminRoutes = require('./routes/admin');
7 const shopRoutes = require('./routes/shop');
8
9 app.use(bodyParser.urlencoded({extended: false}));
10
11 app.use('/admin', adminRoutes);
12 app.use(shopRoutes);
13
14 app.use((req, res, next) => {
15   res.status(404).send('<h1>Page not found</h1>');
16 });
17
18 app.listen(3000);
19
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
{ title: 'Book' }
```

The screenshot shows the same VS Code editor with the '04-expressjs' project. The Explorer sidebar is the same. The main editor displays `app.js` with the following code:

```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.get('/add-product', (req, res, next) => {
6   res.send(
7     '<form action="/admin/add-product" method="POST"><input type="text" name="title"><button ty
8   );
9 });
10
11 router.post('/add-product', (req, res, next) => {
12   console.log(req.body);
13   res.redirect('/');
14 });
15
16 module.exports = router;
17
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
{ title: 'Book' }
```

- so this filtering mechanism here in `app.js` allow us to put a common starting segment for our path which all routes in a given file use to outsource that into this `app.js` so that we don't have to repeat it for all the routes here.

```
//app.js
```

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express()
```

```

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}))

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use((req, res, next) => {
  res.status(404).send('<h1>Page not found</h1>')
})

app.listen(3000);

```

```

//./routes/admin.js

const express = require('express')

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', (req, res, next) => {
  res.send('<form action="/admin/add-product" method="POST"><input type="text"
})

// /admin/add-product => POST
router.post('/add-product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/');
})

module.exports = router;

```

```

//./routes/shop.js

const express = require('express');

const router = express.Router();

router.get('/', (req, res, next) => {
  res.send('<h1>Hello from Express!</h1>')
})

module.exports = router

```

## 🔖\* Chapter 68: Creating HTML Pages

## 1. update

- shop.html
- add-product.html

```
<!--./views/shop.html-->

<!--
    it's a file i wanna serve for users visiting just '/'
-->

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Add Product</title>
</head>

<body>
    <header>
        <nav>
            <ul>
                <li><a href="/">Shop</a></li>
                <li><a href="/admin/add-product">Add Product</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <h1>My Products</h1>
        <p>List of all the products...</p>
    </main>
</body>

</html>
```

```
<!--./views/add-product.html-->

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Add Product</title>
</head>
```

```

<body>
  <header>
    <nav>
      <ul>
        <li><a href="/">Shop</a></li>
        <li><a href="/admin/add-product">Add Product</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <form action="/add-product" method="POST">
      <input type="text" name="title">
      <button type="submit">Add Product</button>
    </form>
  </main>
</body>

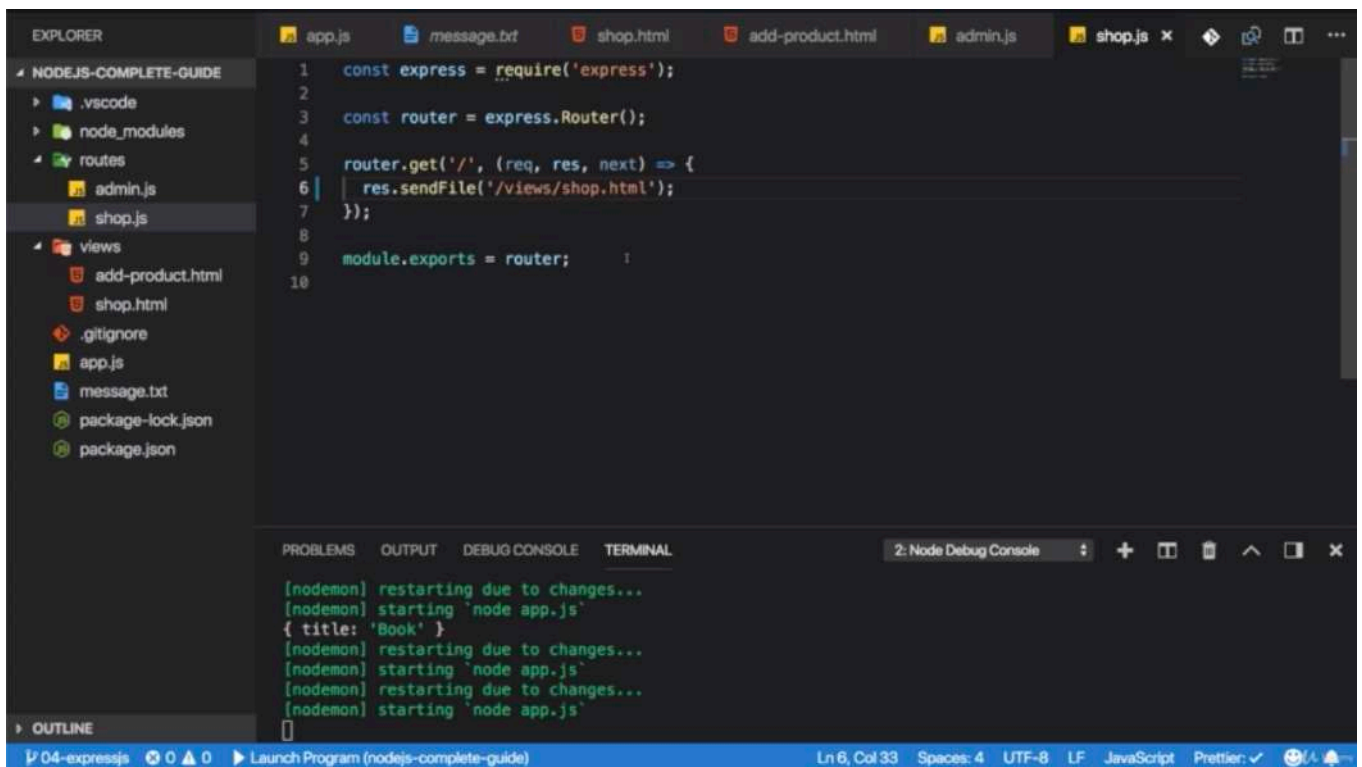
</html>

```

## \* Chapter 69: Serving HTML Pages

### 1. update

- admin.js
- shop.js





Error: ENOENT: no such file or directory, stat '/views/shop.html'

The screenshot shows the Chrome DevTools Network tab. The top section displays two requests: 'localhost' with a status of 404 and 'ng-validate.js' with a status of 200. The 'localhost' request is highlighted in blue. Below the requests, a table shows the details of the requests. The table has columns: Name, Status, Type, Size, and Waterfall. The 'localhost' request is highlighted in blue. The 'ng-validate.js' request is also highlighted in blue. The table shows that the 'localhost' request is a 404 error, while the 'ng-validate.js' request is a successful 200 response. The 'localhost' request is a document type, while the 'ng-validate.js' request is a script type. The 'localhost' request is 449 B in size, while the 'ng-validate.js' request is (from dis...) in size. The 'localhost' request is a document type, while the 'ng-validate.js' request is a script type. The 'localhost' request is 449 B in size, while the 'ng-validate.js' request is (from dis...) in size.

Name	Status	Type	Size	Waterfall
localhost	404	do...	449 B	
ng-validate.js	200	scr...	(from dis...)	

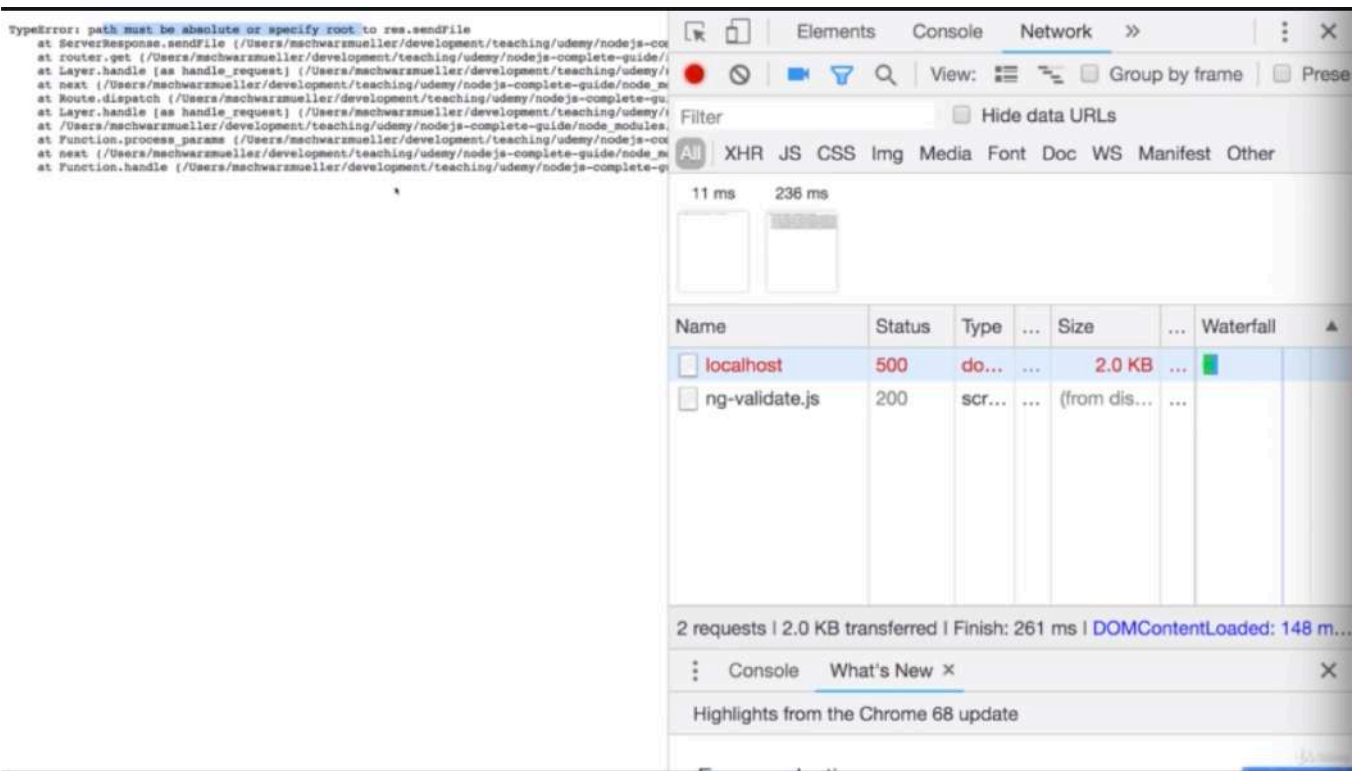
2 requests | 449 B transferred | Finish: 245 ms | DOMContentLoaded: 145 m...

The screenshot shows the VS Code editor with the 'app.js' file open. The file contains the following code:

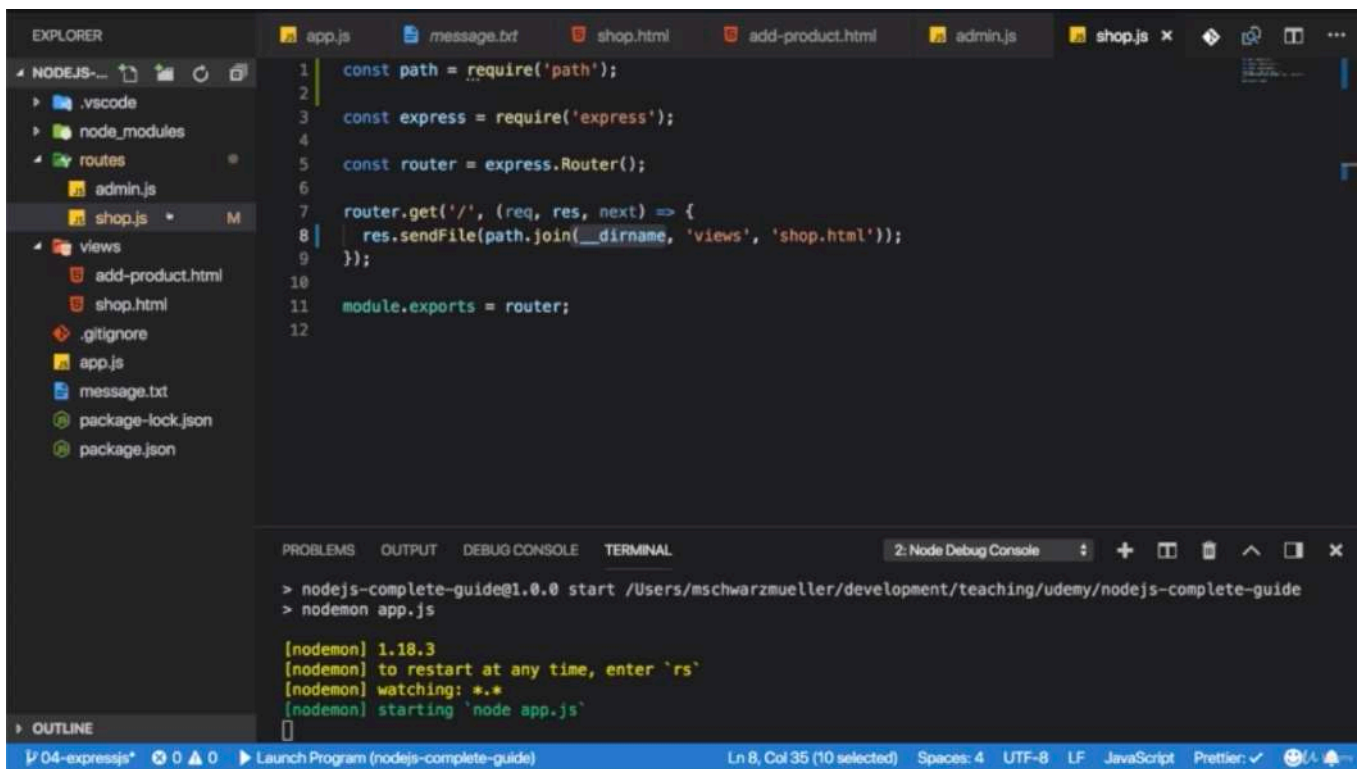
```
1 const express = require('express');
2
3 const router = express.Router();
4
5 router.get('/', (req, res, next) => {
6   res.sendFile('/views/shop.html');
7 });
8
9 module.exports = router;
10
```

The Explorer panel on the left shows the project structure, including the 'views' directory with 'add-product.html' and 'shop.html' files. The Terminal panel at the bottom shows the output of the 'node app.js' command, which includes the error message: 'Error: ENOENT: no such file or directory, stat '/views/shop.html''.

```
[nodemon] starting 'node app.js'
{ title: 'Book' }
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Error: ENOENT: no such file or directory, stat '/views/shop.html'
```



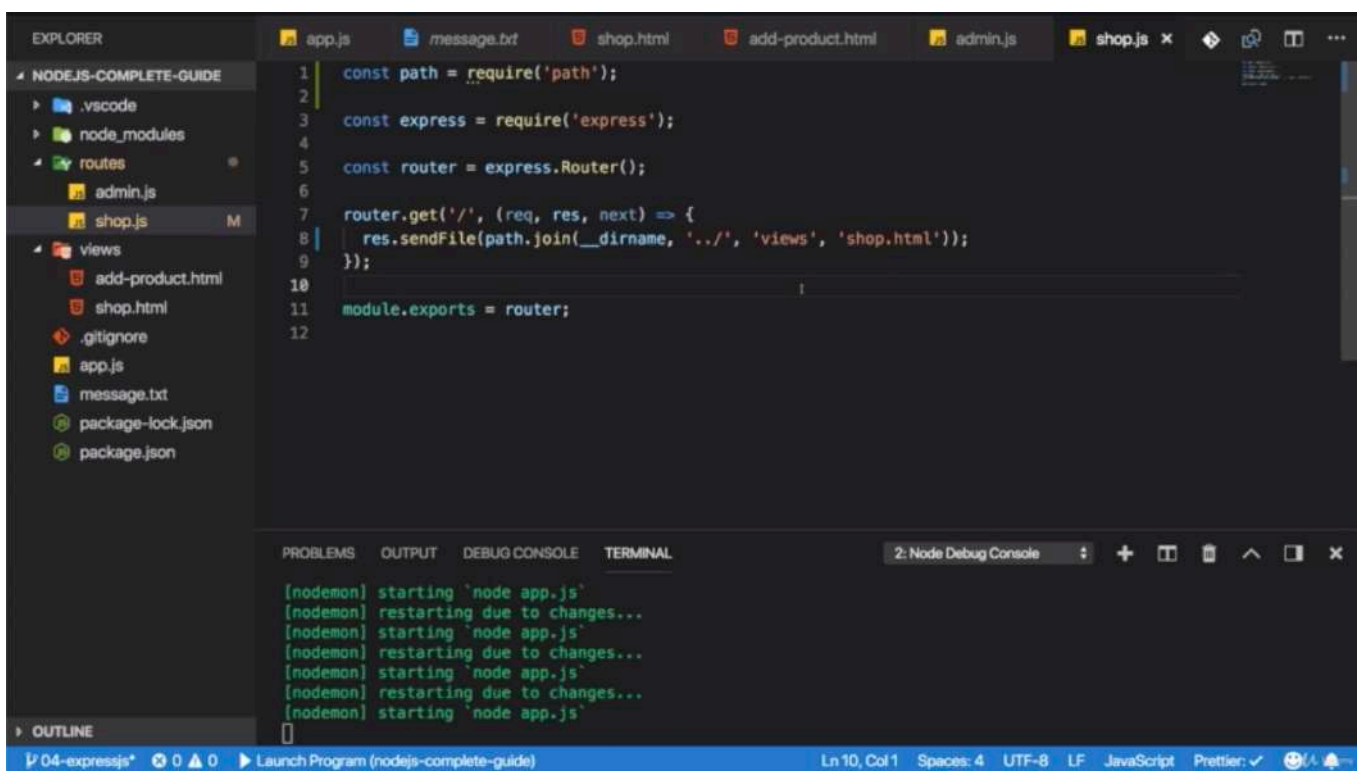
- i don't see anything because this path `'/views/shop.html'` is incorrect. if we now reload, path must be absolute is the error we get.
- absolute path would be correct but slash like this refers to our root folder on our operating system. not the project folder.



- so in order to construct the path to this directory and this file ultimately, we can use a feature provided by node.js called `'path'` the core module.
- `'join()'` yields us a path at the end, it returns a path but it construct this path by

concatenating the different segment.

- the first segment we should pass here is then a global variable made available by node.js and that is '\_\_\_dirname' which is a global variable to holds the absolute path on our operating system. to this project folder and now we can add a , 'views' because the first segment is the path to this whole project folder \_\_\_dirname. and the next segment is that we wanna go into the views folder and then the 3rd segment will be our file. so here shop.html and don't add / because we use path.join() not because of the absolute path, we could build this with \_\_\_dirname and then concatenating this manually too.
- but we are using path.join() because this will automatically build the path in a way that works on both linux and windows systems. on linux, you have paths like '/user/products' but in windows backslash is used like '\\user\\products' adding path.
- therefore if you manually construct this with slashes, it would not run on windows and the other way around.
- path.join() detects operating system you are running on and then automatically build a correct path.

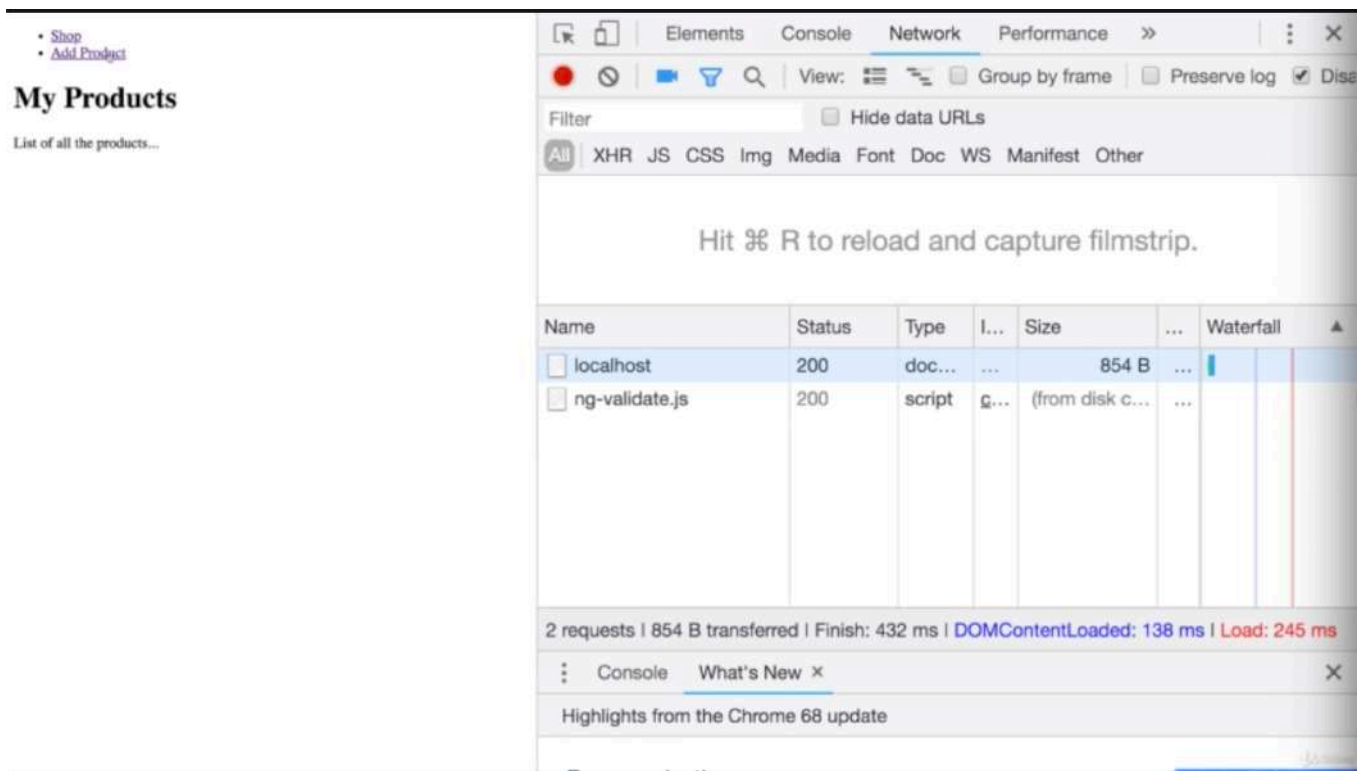


The screenshot shows a VS Code editor with a project named 'NODEJS-COMPLETE-GUIDE'. The Explorer sidebar on the left shows the file structure: .vscode, node\_modules, routes (containing admin.js, shop.js, and views), and views (containing add-product.html, shop.html, .gitignore, app.js, message.txt, package-lock.json, and package.json). The main editor area shows the content of 'shop.js' in the 'routes' folder. The code in 'shop.js' is as follows:

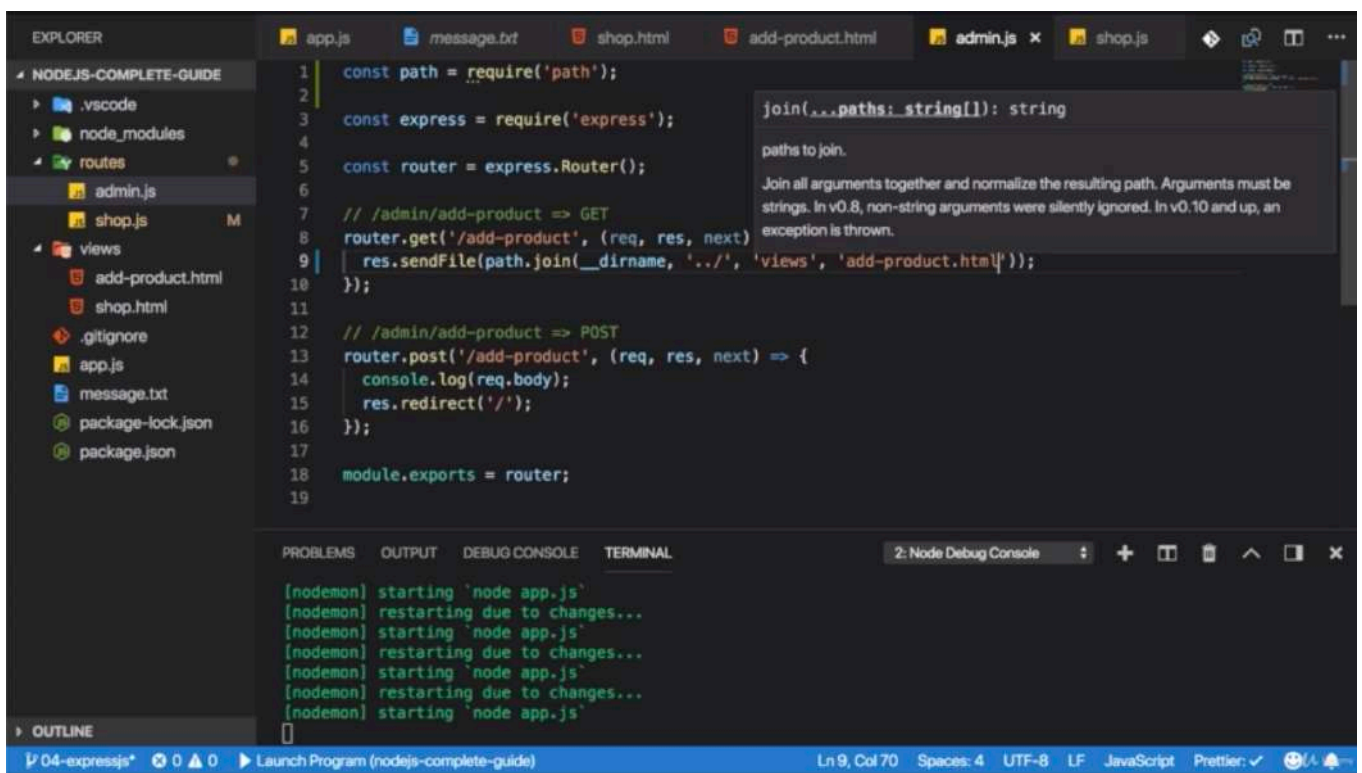
```
1  const path = require('path');
2
3  const express = require('express');
4
5  const router = express.Router();
6
7  router.get('/', (req, res, next) => {
8    res.sendFile(path.join(__dirname, '../', 'views', 'shop.html'));
9  });
10
11
12 module.exports = router;
```

The terminal at the bottom shows the output of running 'node app.js' using nodemon. The output indicates that the application is starting and restarting multiple times due to changes.

- but actually \_\_\_dirname will point routes folder because '\_\_\_dirname' gives us the path to a file in which we use it and we are using it in the shop.js file in the routes folder. but views folder is located in a sibling folder to routes folder.
- so we use '../' and this simply means go up 1 level. so this will now build a path where it first goes into the folder of these files, so into routes folder then it goes up 1 level then into views folder.



- so if we go to 'localhost:3000/' again, we see that html file being served.



- Shop
- Add Product

Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	I...	Size	...	Waterfall	▲
add-product	200	doc...	...	949 B	...		
ng-validate.js	200	script	g...	(from disk c...	...		

2 requests | 949 B transferred | Finish: 229 ms | DOMContentLoaded: 130 ms | Load: 254 ms

Console What's New ✕

Highlights from the Chrome 68 update

- Shop
- Add Product

Hit ⌘ R to reload and capture filmstrip.

Name	×	Headers	Preview	Response	Cookies	Timing
add-...	▼	Response Headers	view source			
ng-v...		Accept-Ranges: bytes Cache-Control: public, max-age=0 Connection: keep-alive Content-Length: 656 <b>Content-Type: text/html; charset=UTF-8</b> Date: Tue, 28 Aug 2018 13:52:27 GMT ETag: W/"290-16580c4dc6f" Last-Modified: Tue, 28 Aug 2018 13:41:49 GMT				

2 request...

Console What's New ✕

Highlights from the Chrome 68 update

- if we go to '/admin/add-product', we see this page too.

```

//./routes/admin.js
const path = require('path')

const express = require('express')

const router = express.Router();

// /admin/add-product => GET

```

```
router.get('/add-product', (req, res, next) => {
  res.sendFile(path.join(__dirname, '../', 'views', 'add-product.html'))
})

// /admin/add-product => POST
router.post('/add-product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/');
})

module.exports = router;
```

```
//./routes/shop.js
const path = require('path');

const express = require('express');

const router = express.Router();

router.get('/', (req, res, next) => {
  res.sendFile(path.join(__dirname, '../', 'views', 'shop.html'))
})

module.exports = router
```

## \* Chapter 70: Returning A 404 Page

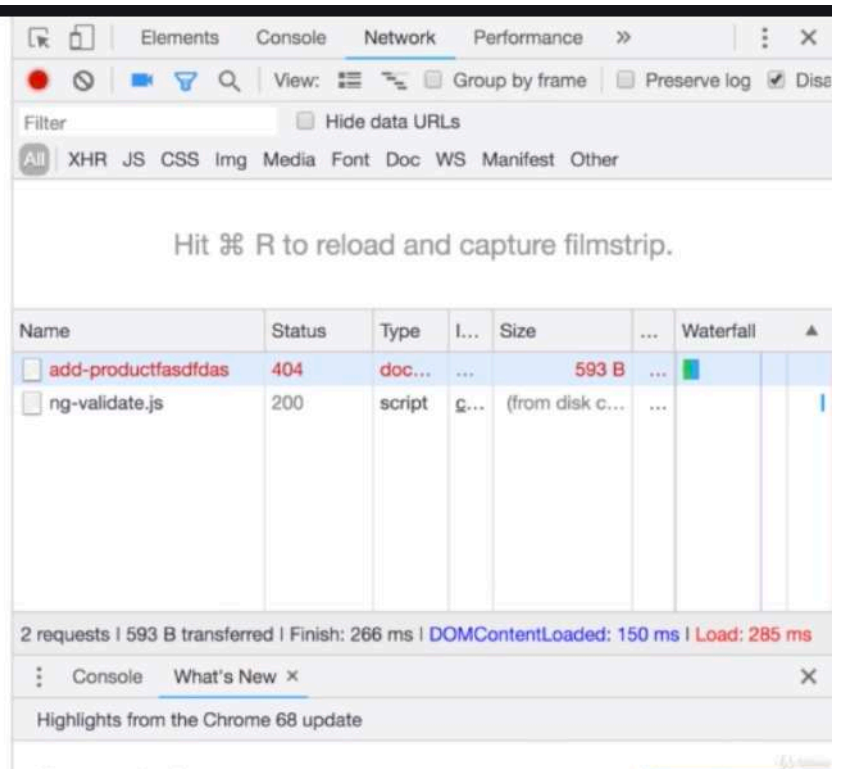
---

### 1. update

- 404.html
- app.js



Page Not Found!



```
//app.js

const path = require('path')

const express = require('express');
const bodyParser = require('body-parser');

const app = express()

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}))

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use((req, res, next) => {
  res.status(404).sendFile(path.join(__dirname, 'views', '404.html'))
})

app.listen(3000);
```

```
<!--./views/404.html-->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

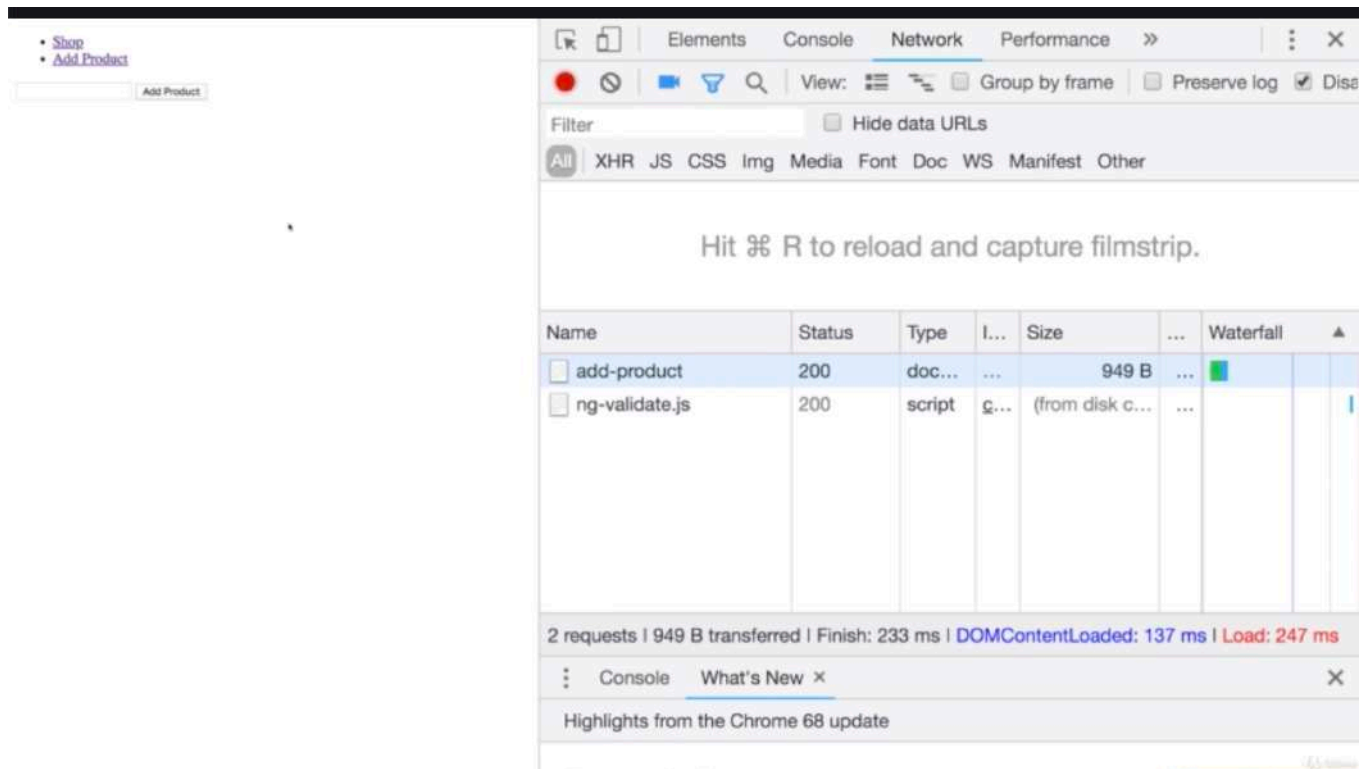


```
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Page Not Found</title>
</head>
<body>
  <h1>Page Not Found</h1>
</body>
</html>
```

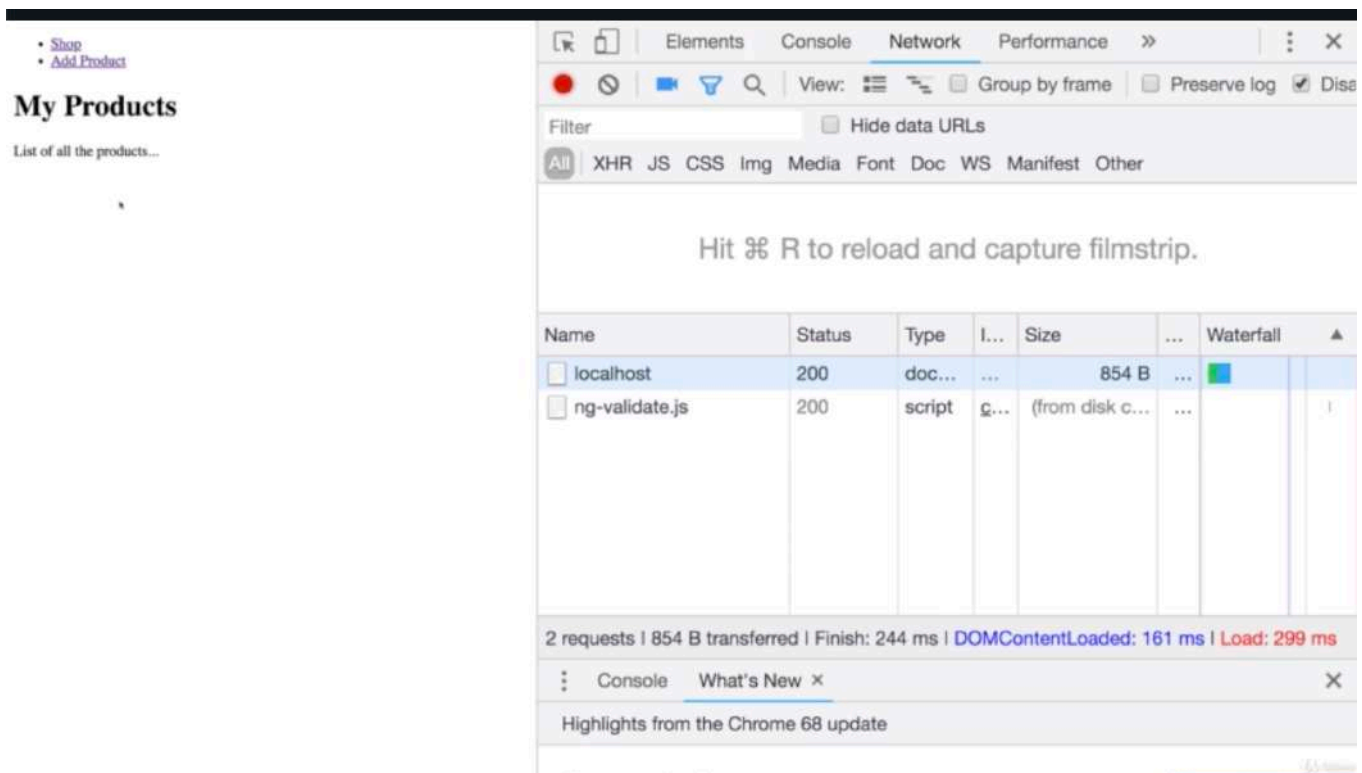
## \* Chapter 71: Using A Helper Function For Navigation

### 1. update

- ./util/path.js
- shop.js
- admin.js



- after reload, it still works. because now we are in the end having a pretty neat way of constructing a path to our root directory.
- i will do the same in shop.js



- you could have stuck to the old approach but this one is a even cleaner one and one that should work on all operating systems and it always gives you the path to the root file.

```
//./routes/shop.js
const path = require('path');

const express = require('express');

const rootDir = require('../util/path')

const router = express.Router();

router.get('/', (req, res, next) => {
  res.sendFile(path.join(rootDir, 'views', 'shop.html'))
})

module.exports = router
```

```
//./routes/admin.js
const path = require('path')

const express = require('express')

const rootDir = require('../util/path')

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', (req, res, next) => {
```

```

    res.sendFile(path.join(rootDir, 'views', 'add-product.html'))
  })

// /admin/add-product => POST
router.post('/add-product', (req, res, next) => {
  console.log(req.body)
  res.redirect('/');
})

module.exports = router;

```

```

//./util/path.js

const path = require('path')

/**if we use that
 * we just have to find out
 * which directory or for which file we wanna get the directory name
 *
 * there we can use the global 'process' variable that is also a variable that is
 * you don't need to import it
 * and there you will have a mainModule property.
 * this will refer to the main module that started your application
 * so to the module we created in app.js
 * and now we can call file name to find out in which file this module was spun u
 *
 * so 'process.mainModule.filename' gives us the path to the file
 * that is responsible for the fact that our application is running
 * and filename is what we put into dirname to get a path to that directory.
 * */
module.exports = path.dirname(process.mainModule.filename)

```

## 🔗 \* Chapter 72: Styling Our Pages

### 1. update

- shop.html
- add-product.html
- 404.html

```

<!--shop.html-->

<!DOCTYPE html>
<html lang="en">

<head>

```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Add Product</title>
<style>
  body {
    padding: 0;
    margin: 0;
    font-family: sans-serif;
  }

  main {
    padding: 1rem;
  }

  .main-header {
    width: 100%;
    height: 3.5rem;
    background-color: #dbc441;
    padding: 0 1.5rem;
  }

  .main-header__nav {
    height: 100%;
    display: flex;
    align-items: center;
  }

  .main-header__item-list {
    list-style: none;
    margin: 0;
    padding: 0;
    display: flex;
  }

  .main-header__item {
    margin: 0 1rem;
    padding: 0;
  }

  .main-header__item a {
    text-decoration: none;
    color: black;
  }

  .main-header__item a:hover,
  .main-header__item a:active,
  .main-header__item a.active {
    color: #3e00a1;
  }
</style>
</head>

<body>
```

```

<header class="main-header">
  <nav class="main-header__nav">
    <ul class="main-header__item-list">
      <li class="main-header__item"><a class="active" href="/">Shop</a>
      <li class="main-header__item"><a href="/admin/add-product">Add Pro
    </ul>
  </nav>
</header>

<main>
  <h1>My Products</h1>
  <p>List of all the products...</p>
</main>
</body>

</html>

```

```

<!--add-product.html-->

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Add Product</title>
  <style>
    body {
      padding: 0;
      margin: 0;
      font-family: sans-serif;
    }

    main {
      padding: 1rem;
    }

    .main-header {
      width: 100%;
      height: 3.5rem;
      background-color: #dbc441;
      padding: 0 1.5rem;
    }

    .main-header__nav {
      height: 100%;
      display: flex;
      align-items: center;
    }

    .main-header__item-list {

```

```
list-style: none;
margin: 0;
padding: 0;
display: flex;
}

.main-header__item {
margin: 0 1rem;
padding: 0;
}

.main-header__item a {
text-decoration: none;
color: black;
}

.main-header__item a:hover,
.main-header__item a:active,
.main-header__item a.active {
color: #3e00a1;
}

.product-form {
width: 20rem;
max-width: 90%;
margin: auto;
}

.form-control {
margin: 1rem 0;
}

.form-control label,
.form-control input {
display: block;
width: 100%;
}

.form-control input {
border: 1px solid #dbc441;
font: inherit;
border-radius: 2px;
}

button {
font: inherit;
border: 1px solid #3e00a1;
color: #3e00a1;
background: white;
border-radius: 3px;
cursor: pointer;
}

button:hover,
```

```

        button:active {
            background-color: #3e00a1;
            color: white;
        }
    </style>
</head>

<body>
    <header class="main-header">
        <nav class="main-header__nav">
            <ul class="main-header__item-list">
                <li class="main-header__item"><a href="/">Shop</a></li>
                <li class="main-header__item"><a class="active" href="/admin/add-
            </ul>
        </nav>
    </header>

    <main>
        <form class="product-form" action="/admin/add-product" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input type="text" name="title" id="title">

                <button type="submit">Add Product</button>
            </form>
        </main>
    </body>

</html>

```

```

<!--./views/404.html-->

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Page Not Found</title>
    <style>
        body {
            padding: 0;
            margin: 0;
            font-family: sans-serif;
        }

        main {
            padding: 1rem;
        }
    </style>

```



```

    .main-header {
        width: 100%;
        height: 3.5rem;
        background-color: #dbc441;
        padding: 0 1.5rem;
    }

    .main-header__nav {
        height: 100%;
        display: flex;
        align-items: center;
    }

    .main-header__item-list {
        list-style: none;
        margin: 0;
        padding: 0;
        display: flex;
    }

    .main-header__item {
        margin: 0 1rem;
        padding: 0;
    }

    .main-header__item a {
        text-decoration: none;
        color: black;
    }

    .main-header__item a:hover,
    .main-header__item a:active,
    .main-header__item a.active {
        color: #3e00a1;
    }
</style>
</head>

<body>
    <header class="main-header">
        <nav class="main-header__nav">
            <ul class="main-header__item-list">
                <li class="main-header__item"><a class="active" href="/">Shop</a>
                <li class="main-header__item"><a href="/admin/add-product">Add Pro
            </ul>
        </nav>
    </header>
    <h1>Page Not Found!</h1>
</body>

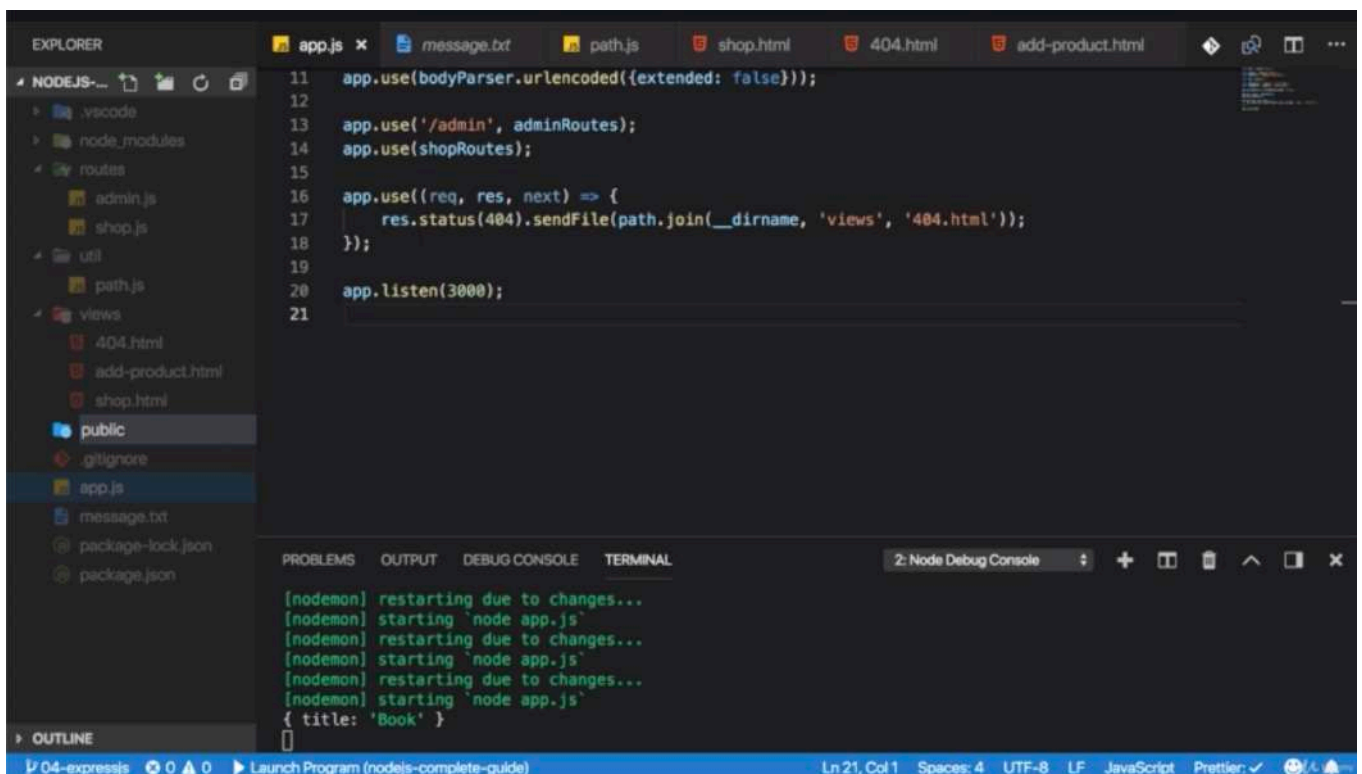
</html>

```

## \* Chapter 73: Serving Files Statically

### 1. update

- app.js
- ./views/shop.html
- ./views/add-product.html
- ./views/404.html
- ./public/css/main.css
- ./public/css/product.css



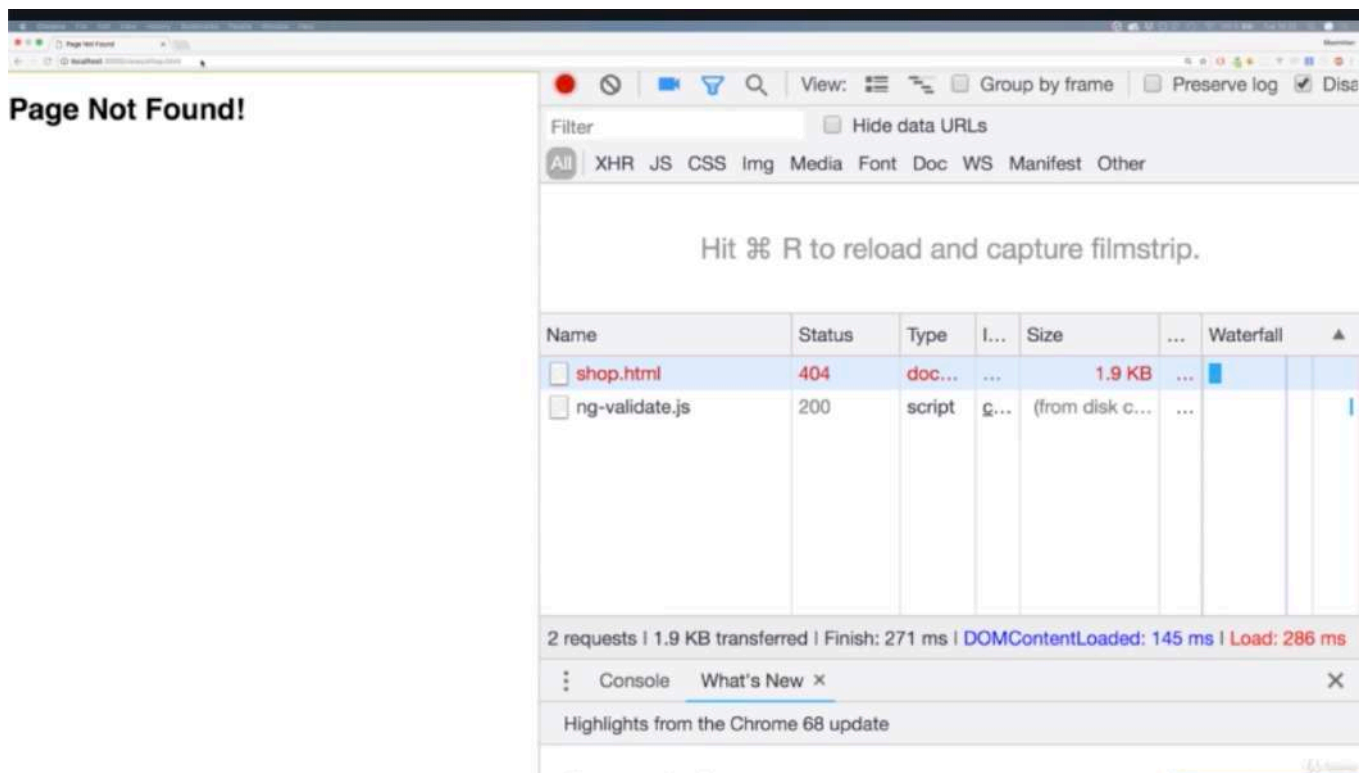
The screenshot shows the Visual Studio Code editor with the following components:

- EXPLORER:** A file tree on the left showing the project structure. The 'public' folder is selected.
- EDITOR:** The main workspace showing the `app.js` file. The code is as follows:

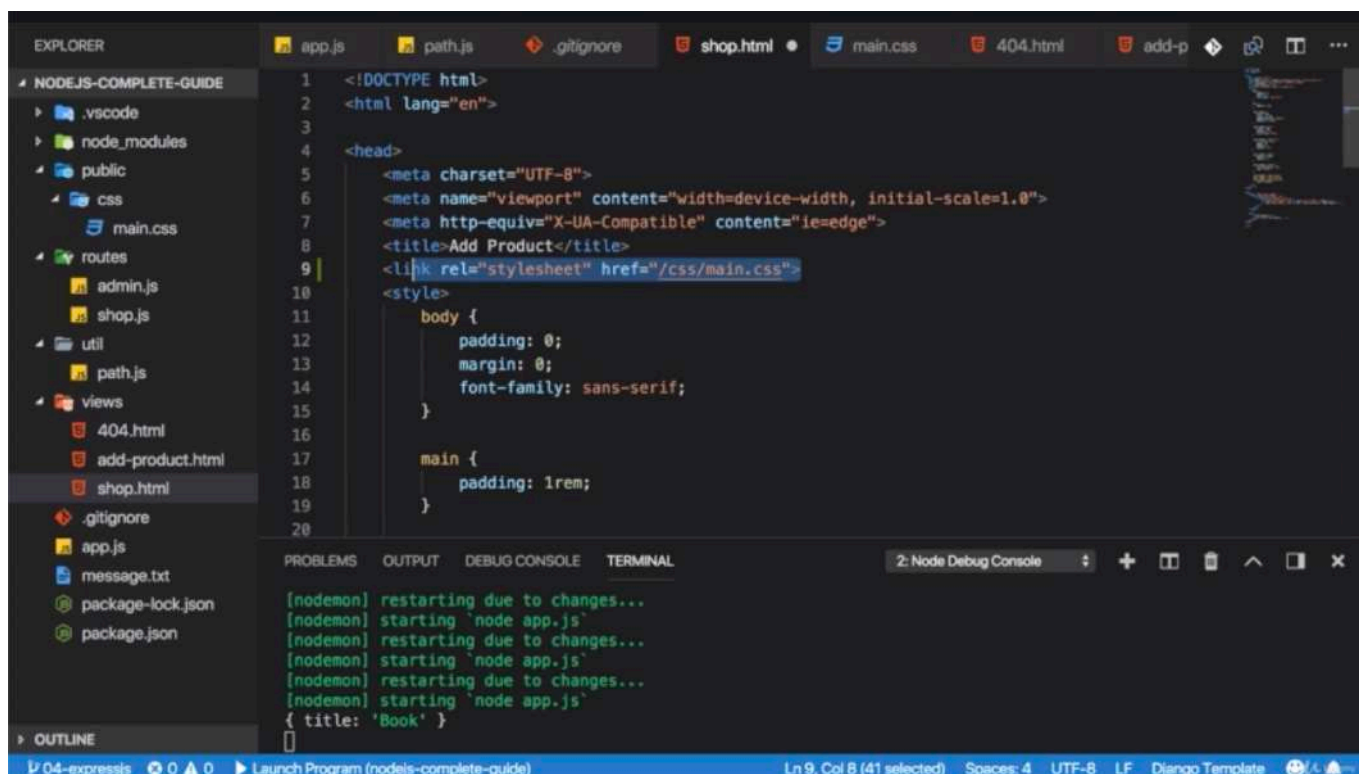
```
11 app.use(bodyParser.urlencoded({extended: false}));
12
13 app.use('/admin', adminRoutes);
14 app.use(shopRoutes);
15
16 app.use((req, res, next) => {
17   res.status(404).sendFile(path.join(__dirname, 'views', '404.html'));
18 });
19
20 app.listen(3000);
21
```
- TERMINAL:** The bottom panel shows the output of the Node.js application. It displays several messages from nodemon, indicating that the application is restarting due to changes and then starting successfully. The final output is:

```
{ title: 'Book' }
```

- but the convention is to call it 'public' because you wanna indicate that this is a folder that holds content which are always exposed to the public crowd or which is always exposed to the public. so where you don't need any permissions to access and that's important.



- all your files are not accessible by your users. if you ever tried to enter localhost and then something like views, shop.html, that will not work because this is simply accepted by express and it tries to find a route that matches this. it tries to find it here in app.js. and also in shop routes and so on. it doesn't find that route and therefore it doesn't give you access, you can't access the file system here through URL



- but now i wanna make an exception. i want the some requests can access the file system because let's say in shop.html, i wanna have something like a link here where i point at something like main.css anything like that.

- and my imagination would be that in public folder, i have a css folder with main css file. that's the file i wanna serve with this link.

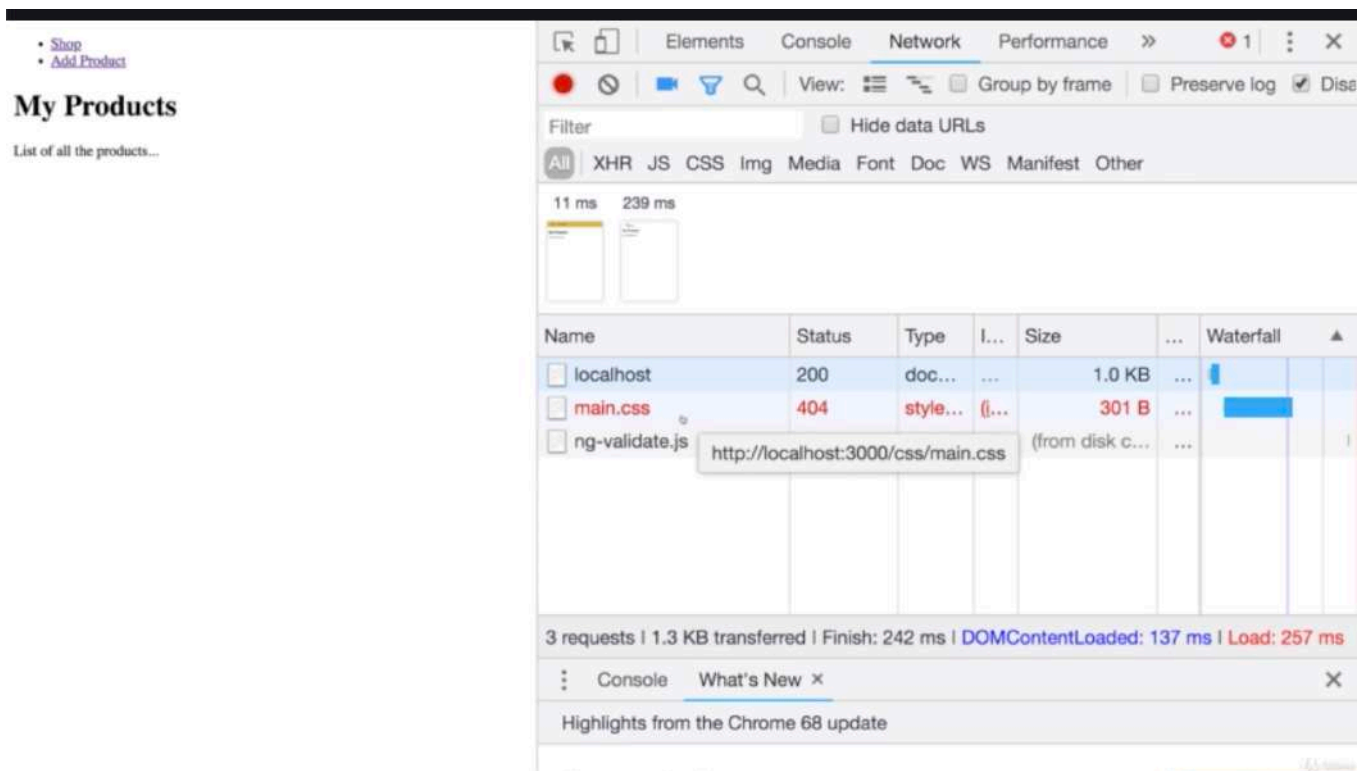
The screenshot shows a VS Code editor with a project named 'NODEJS-COMPLETE-GUIDE'. The file explorer on the left shows a directory structure with 'public' containing a 'css' folder with 'main.css'. The main editor displays the content of 'main.css':

```

1  body {
2    padding: 0;
3    margin: 0;
4    font-family: sans-serif;
5  }
6
7  main {
8    padding: 1rem;
9  }
10
11 .main-header {
12   width: 100%;
13   height: 3.5rem;
14   background-color: #dbc441;
15   padding: 0 1.5rem;
16 }
17
18 .main-header__nav {
19   height: 100%;
20   display: flex;

```

The terminal at the bottom shows a series of 'nodemon' restarts, indicating the application is running and being monitored for changes.



- if i save and reload my main page, all the styling is gone because it can't find the main css file as far as you can see here in the developer tools because we can't access the file system.
- path is incorrect. it's public/css.

```
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Add Product</title>
9   <link rel="stylesheet" href="/public/css/main.css">
10 </head>
11
12 <body>
13   <header class="main-header">
14     <nav class="main-header__nav">
15       <ul class="main-header__item-list">
16         <li class="main-header__item"><a class="active" href="/">Shop</a></li>
17         <li class="main-header__item"><a href="/admin/add-product">Add Product</a></li>
18       </ul>
19     </nav>
20   </header>
21
22   <main>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

2: Node Debug Console

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
{ title: 'Book' }
```

## My Products

List of all the products...

Hit ⌘ R to reload and capture filmstrip.

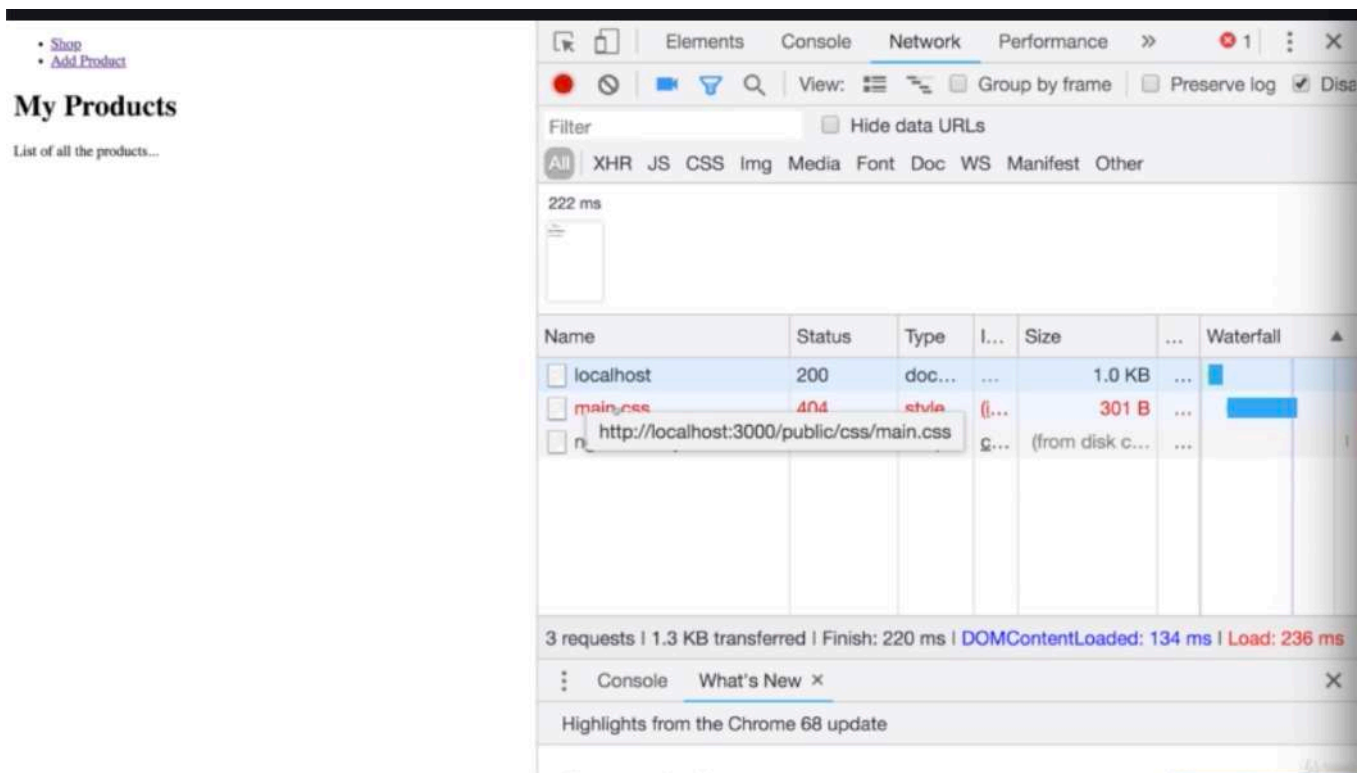
Name	Status	Type	I...	Size	...	Waterfall	▲
localhost	200	doc...	...	1.0 KB	...		
main.css	404	style...	(l...	301 B	...		
ng-validate.js	200	script	g...	(from disk c...	...		

3 requests | 1.3 KB transferred | Finish: 251 ms | DOMContentLoaded: 126 ms | Load: 273 ms

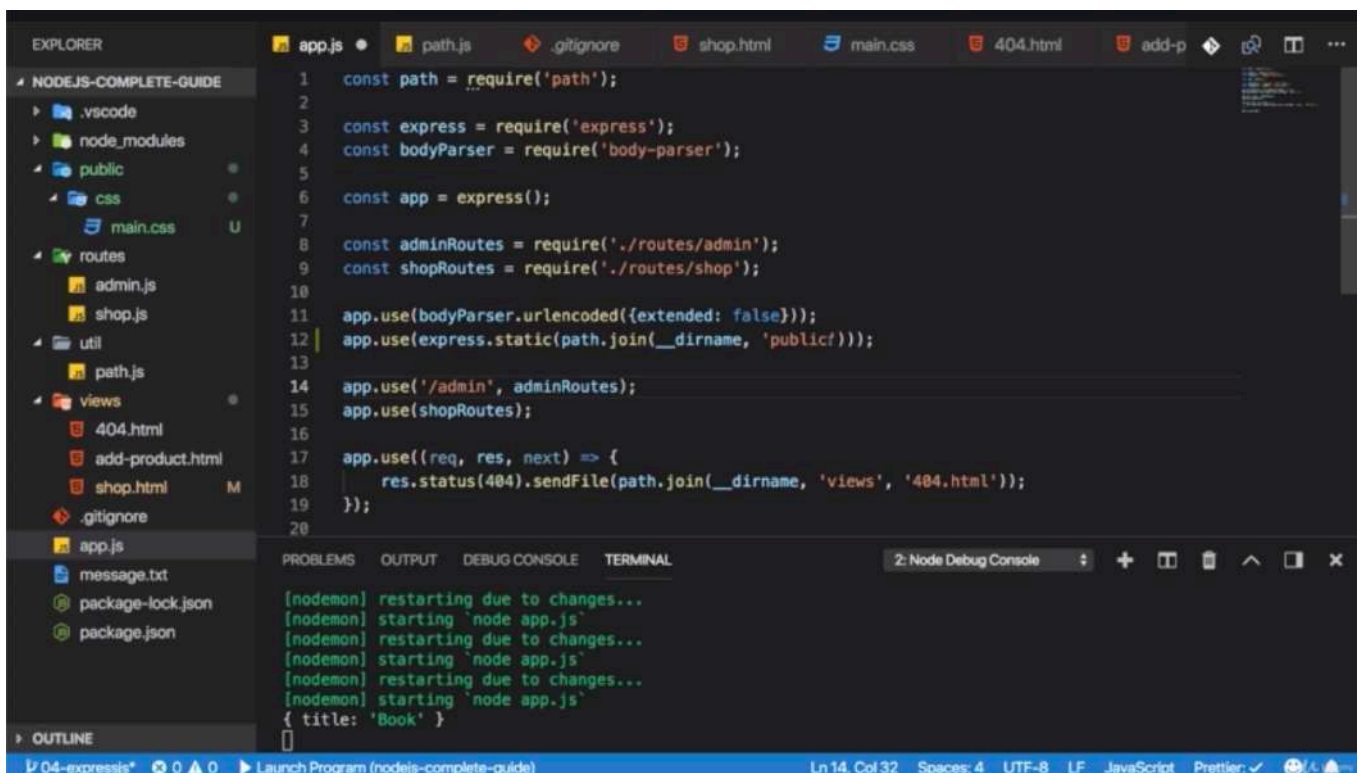
Console What's New

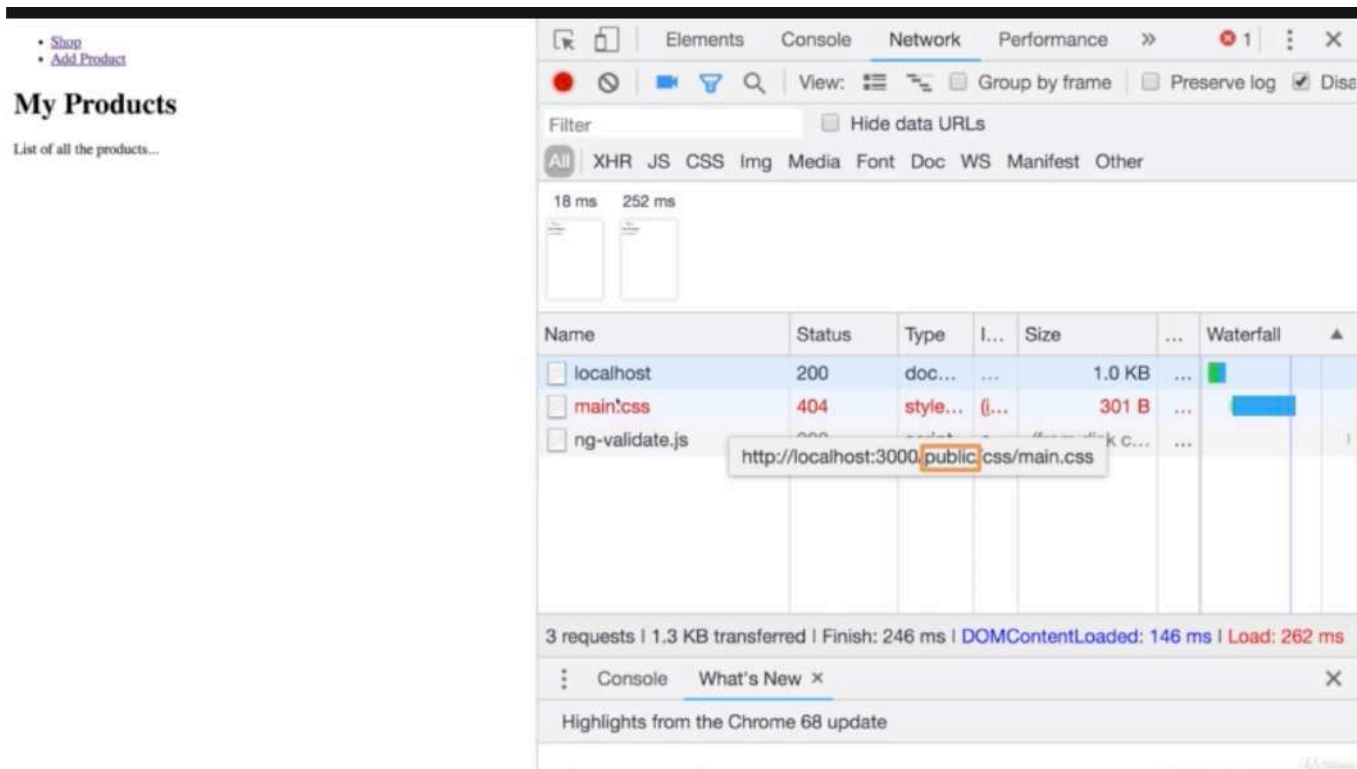
Highlights from the Chrome 68 update



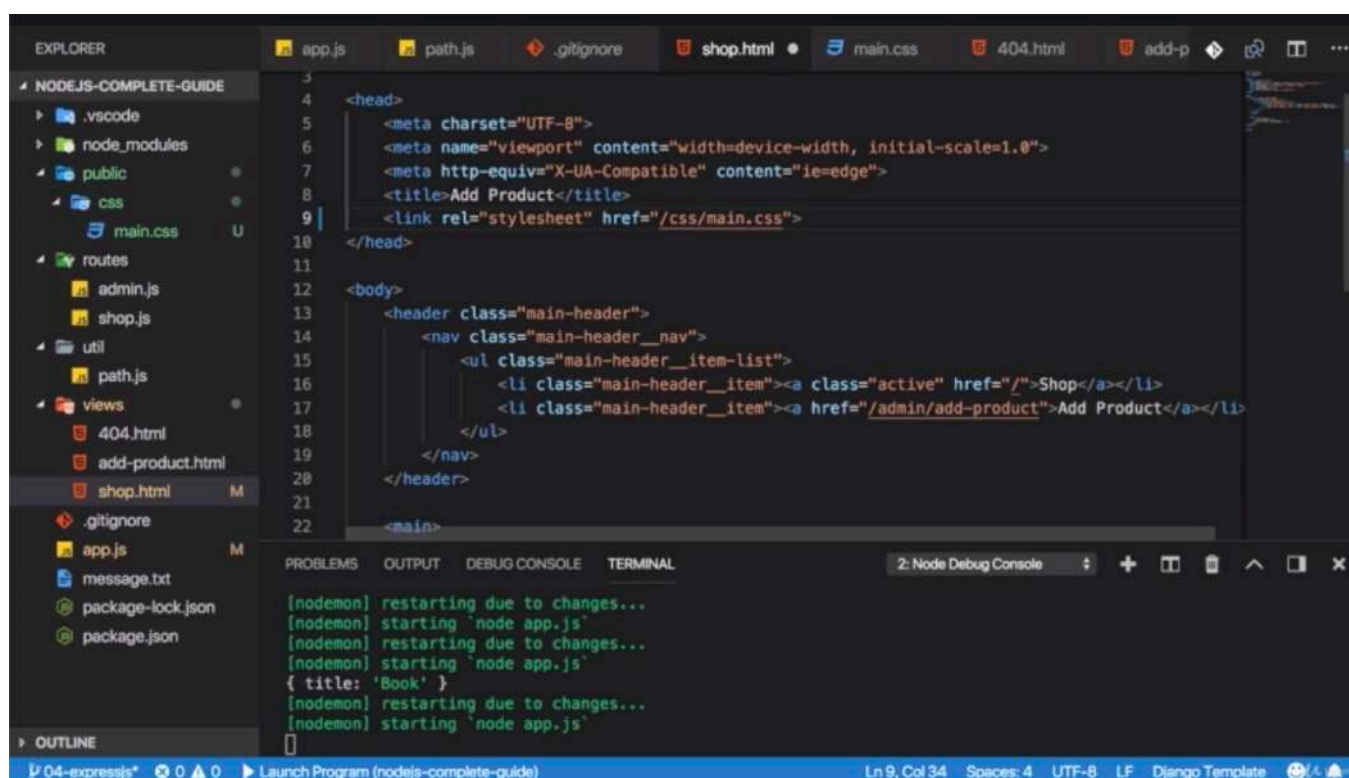


- if i change like that and reload and you will see it never work. and now it does look in the public folder.
- for this, we need a feature express.js offers us. we need to be able to serve files statically and statically simply means not handled by the express router or other middleware. but instead directly forwarded to the file system





- still doesn't work because the path with public at the beginning is wrong.





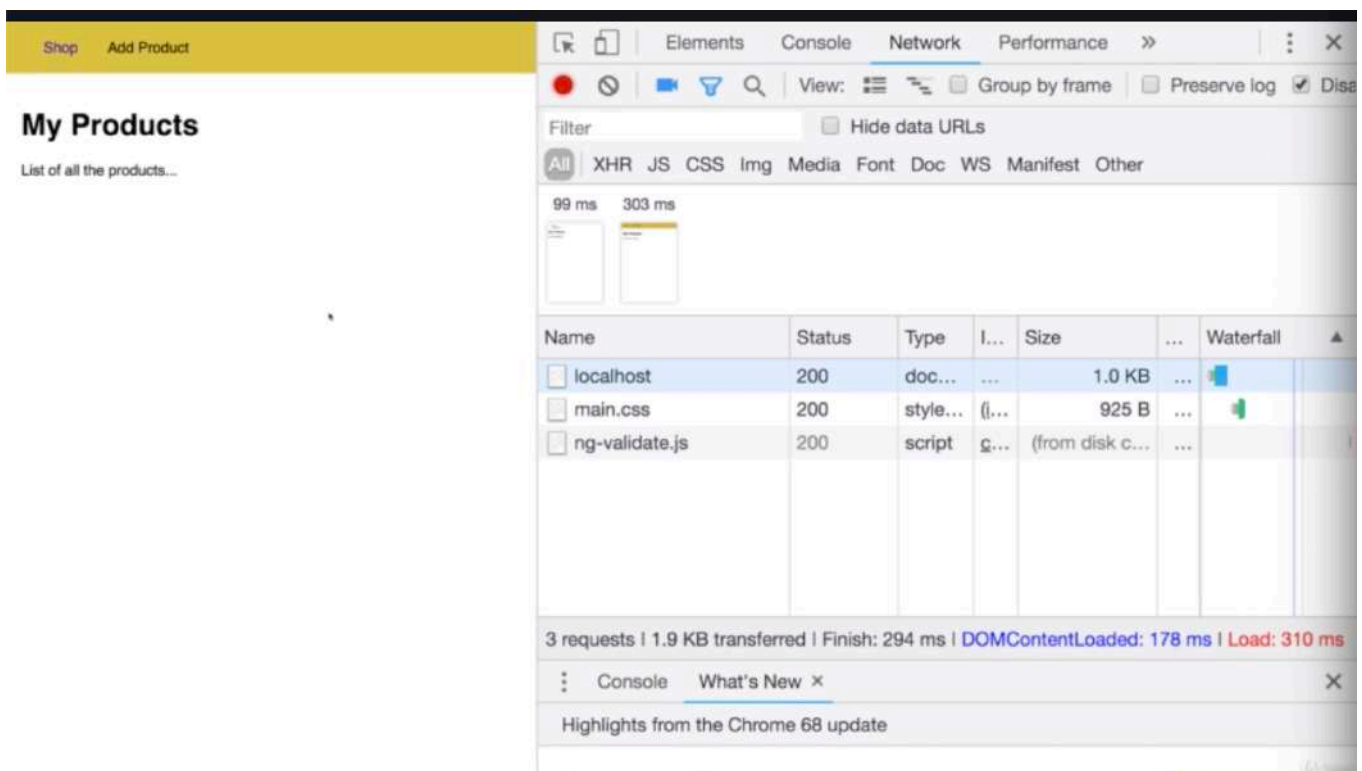
The screenshot shows the VS Code editor with the `app.js` file open. The file contains the following code:

```
1 const path = require('path');
2
3 const express = require('express');
4 const bodyParser = require('body-parser');
5
6 const app = express();
7
8 const adminRoutes = require('./routes/admin');
9 const shopRoutes = require('./routes/shop');
10
11 app.use(bodyParser.urlencoded({extended: false}));
12 app.use(express.static(path.join(__dirname, 'public')));
13
14 app.use('/admin', adminRoutes);
15 app.use(shopRoutes);
16
17 app.use((req, res, next) => {
18   res.status(404).sendFile(path.join(__dirname, 'views', '404.html'));
19 });
20
```

The terminal output shows the following messages:

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
{ title: 'Book' }
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- we should omit this and directly act as if we are in the public folder already. because this is basically what express will do here.
- it will take any request that tries to find some file. so anything that tries to find a .css or .js files, if we have such a request, it automatically forwards it to the public folder
- and therefore then the remaining path has to be everything but that public.



ShopAdd Product

## My Products

List of all the products...

ElementsConsoleNetworkPerformance

View:Group by framePreserve logDiscard

FilterHide data URLs

AllXHRJS CSS Img Media Font Doc WS Manifest Other

Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	I...	Size	...	Waterfall	▲
localhost	200	doc...	...	1.0 KB	...		
main.css	200	style...	(i...	925 B	...		
ng-validate.js	200	script	g...	(from disk c...	...		

3 requests | 1.9 KB transferred | Finish: 244 ms | DOMContentLoaded: 146 ms | Load: 259 ms

ConsoleWhat's New

Highlights from the Chrome 68 update

ShopAdd Product

Title

Add Product

ElementsConsoleNetworkPerformance

View:Group by framePreserve logDiscard

FilterHide data URLs

AllXHRJS CSS Img Media Font Doc WS Manifest Other

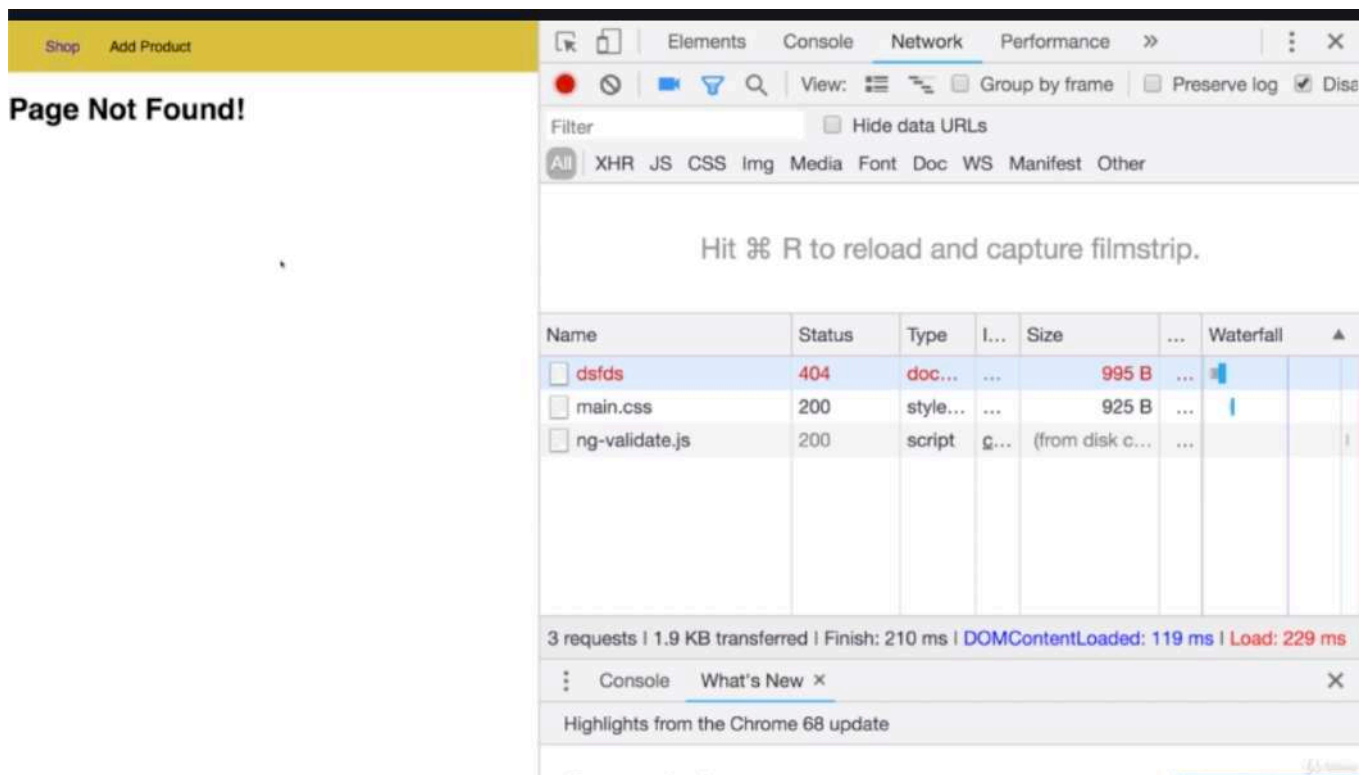
Hit ⌘ R to reload and capture filmstrip.

Name	Status	Type	I...	Size	...	Waterfall	▲
add-product	200	doc...	...	1.3 KB	...		
main.css	200	style...	a...	925 B	...		
product.css	200	style...	a...	818 B	...		
ng-validate.js	200	script	g...	(from disk c...	...		

4 requests | 3.0 KB transferred | Finish: 232 ms | DOMContentLoaded: 127 ms | Load: 246 ms

ConsoleWhat's New

Highlights from the Chrome 68 update



```
//app.js

const path = require('path')

const express = require('express');
const bodyParser = require('body-parser');

const app = express()

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}))
/**'static' is built-in method and this is a built-in middleware
 * it serve static files. so we can execute this function.
 *
 * we have to pass in a path to the folder which we wanna serve statically
 * so a folder which we wanna grant read access to.
 *
 * with this, user should be able to access the public path
 */

/**this could register multiple static folders
 * and it will funnel the request through all of them until it has a first hit f
app.use(express.static(path.join(__dirname, 'public'))))

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use((req, res, next) => {
  res.status(404).sendFile(path.join(__dirname, 'views', '404.html'))
})
```

```
app.listen(3000);
```

```
<!--./views/shop.html-->

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Add Product</title>
  <link rel="stylesheet" href="/css/main.css">
</head>

<body>
  <header class="main-header">
    <nav class="main-header__nav">
      <ul class="main-header__item-list">
        <li class="main-header__item"><a class="active" href="/">Shop</a>
        <li class="main-header__item"><a href="/admin/add-product">Add Pro
      </ul>
    </nav>
  </header>

  <main>
    <h1>My Products</h1>
    <p>List of all the products...</p>
  </main>
</body>

</html>
```

```
<!--./views/add-product.html-->

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Add Product</title>
  <link rel="stylesheet" href="/css/main.css">
  <link rel="stylesheet" href="/css/product.css">
</head>

<body>
  <header class="main-header">
    <nav class="main-header__nav">
```

```

        <ul class="main-header__item-list">
            <li class="main-header__item"><a href="/">Shop</a></li>
            <li class="main-header__item"><a class="active" href="/admin/add-product">Add Product</a></li>
        </ul>
    </nav>
</header>

<main>
    <form class="product-form" action="/admin/add-product" method="POST">
        <div class="form-control">
            <label for="title">Title</label>
            <input type="text" name="title" id="title">
        </div>

        <button type="submit">Add Product</button>
    </form>
</main>
</body>

</html>

```

```

<!--./views/404.html-->

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Page Not Found</title>
    <link rel="stylesheet" href="/css/main.css">
</head>

<body>
    <header class="main-header">
        <nav class="main-header__nav">
            <ul class="main-header__item-list">
                <li class="main-header__item"><a class="active" href="/">Shop</a></li>
                <li class="main-header__item"><a href="/admin/add-product">Add Product</a></li>
            </ul>
        </nav>
    </header>
    <h1>Page Not Found!</h1>
</body>

</html>

```

```

/*./public/css/main.css*/

```

```

body {

```

```
padding: 0;
margin: 0;
font-family: sans-serif;
}

main {
padding: 1rem;
}

.main-header {
width: 100%;
height: 3.5rem;
background-color: #dbc441;
padding: 0 1.5rem;
}

.main-header__nav {
height: 100%;
display: flex;
align-items: center;
}

.main-header__item-list {
list-style: none;
margin: 0;
padding: 0;
display: flex;
}

.main-header__item {
margin: 0 1rem;
padding: 0;
}

.main-header__item a {
text-decoration: none;
color: black;
}

.main-header__item a:hover,
.main-header__item a:active,
.main-header__item a.active {
color: #3e00a1;
}
```

```
/*./public/css/product.css*/
```

```
body {
padding: 0;
margin: 0;
font-family: sans-serif;
}
```

```
main {
  padding: 1rem;
}

.main-header {
  width: 100%;
  height: 3.5rem;
  background-color: #dbc441;
  padding: 0 1.5rem;
}

.main-header__nav {
  height: 100%;
  display: flex;
  align-items: center;
}

.main-header__item-list {
  list-style: none;
  margin: 0;
  padding: 0;
  display: flex;
}

.main-header__item {
  margin: 0 1rem;
  padding: 0;
}

.main-header__item a {
  text-decoration: none;
  color: black;
}

.main-header__item a:hover,
.main-header__item a:active,
.main-header__item a.active {
  color: #3e00a1;
}

.product-form {
  width: 20rem;
  max-width: 90%;
  margin: auto;
}

.form-control {
  margin: 1rem 0;
}

.form-control label,
.form-control input {
  display: block;
```



```

width: 100%;
}

.form-control input {
border: 1px solid #dbc441;
font: inherit;
border-radius: 2px;
}

button {
font: inherit;
border: 1px solid #3e00a1;
color: #3e00a1;
background: white;
border-radius: 3px;
cursor: pointer;
}

button:hover,
button:active {
background-color: #3e00a1;
color: white;
}

```

## \* Chapter 74: Wrap Up



### Module Summary

#### What is Express.js?

- Express.js is Node.js framework – a package that adds a bunch of utility functions and tools and a clear set of rules on how the app should be built (middleware!)
- It's highly extensible and other packages can be plugged into it (middleware!)

#### Routing

- You can filter requests by path and method
- If you filter by method, paths are matched exactly, otherwise, the first segment of a URL is matched
- You can use the `express.Router` to split your routes across files elegantly

#### Middleware, `next()` and `res()`

- Express.js relies heavily on middleware functions – you can easily add them by calling `use()`
- Middleware functions handle a request and should call `next()` to forward the request to the next function in line or send a response

#### Serve Files

- You're not limited to serving dummy text as a response
- You can `sendFile()`s to your users – e.g. HTML files
- If a request is directly made for a file (e.g. a `.css` file is requested), you can enable static serving for such files via `express.static()`