Day 46

“Web Development + Security”

Middleware in Express Js:

What is Middleware?

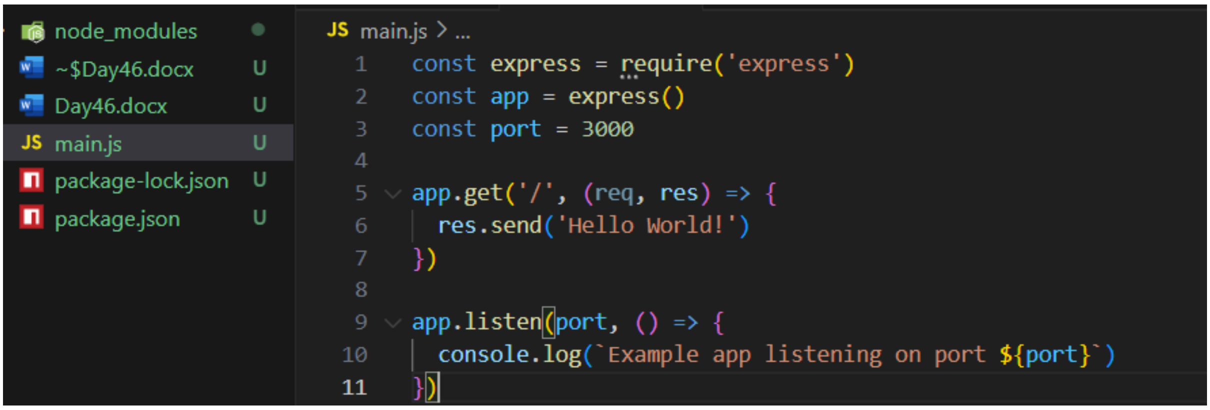
A middleware is a function that runs during the request-response cycle in Express.js. It has access to:

- req → the request object
- res → the response object
- next() → a function to pass control to the next middleware or route handler

Think of it like a pipeline: every request passes through middleware before reaching the final route handler.

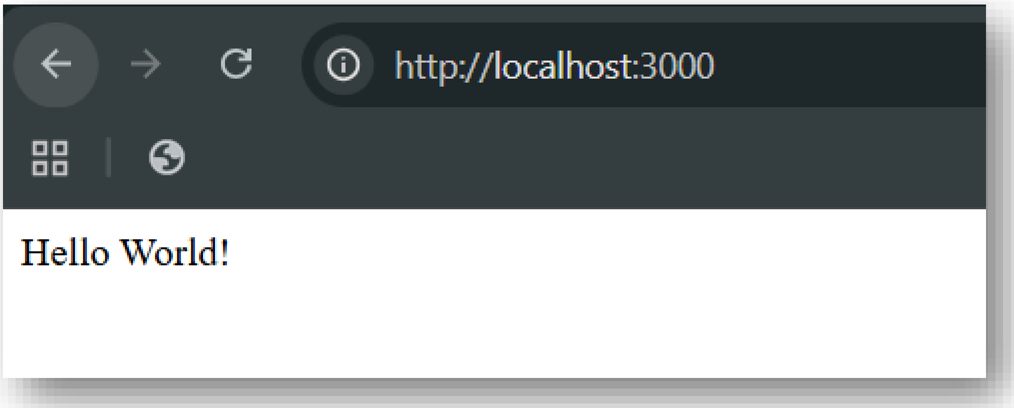
A general template to create:

Main.js:



```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

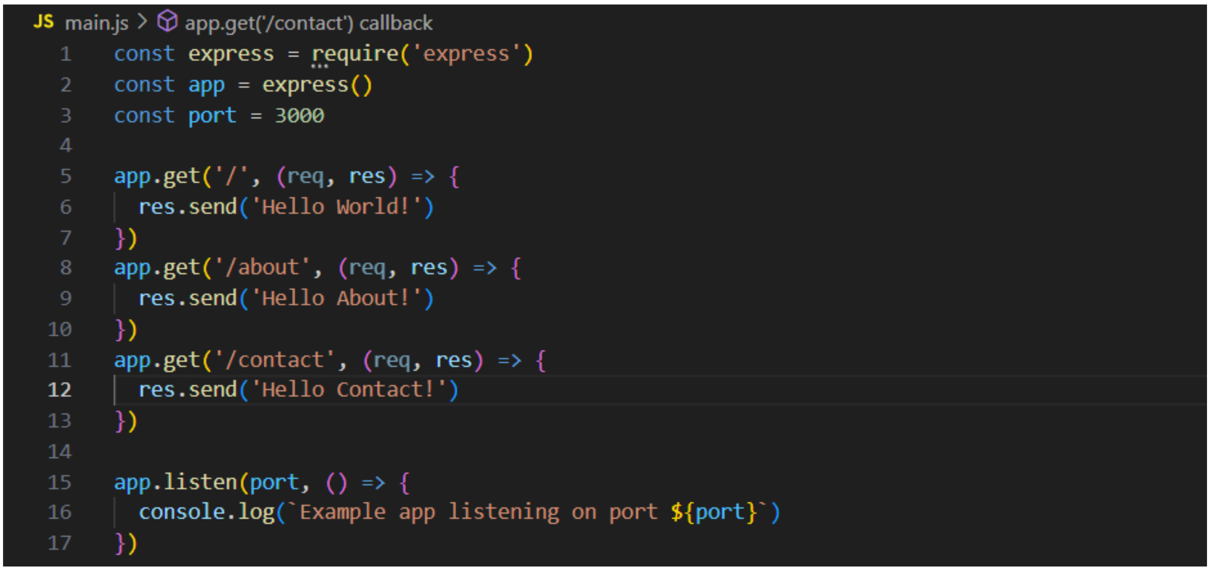
Output:



Hello World!

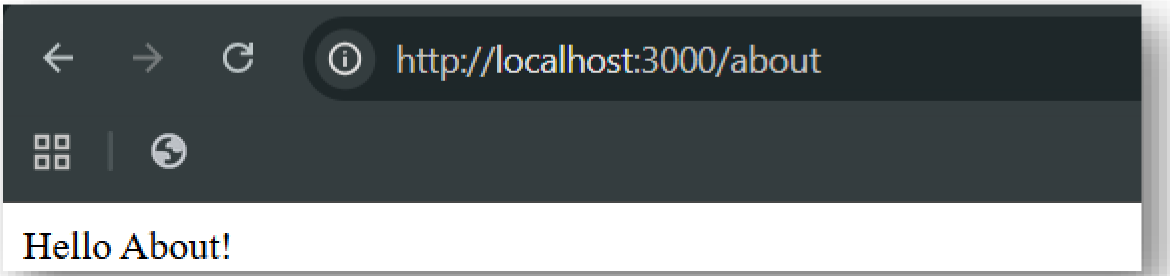
Also, we can create something like this as well:

Main.js:



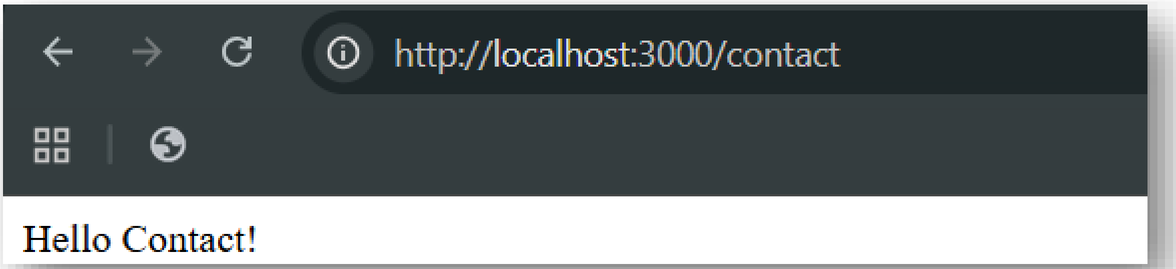
```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Output: at “/about”



Hello About!

Output: at “/contact”



Hello Contact!

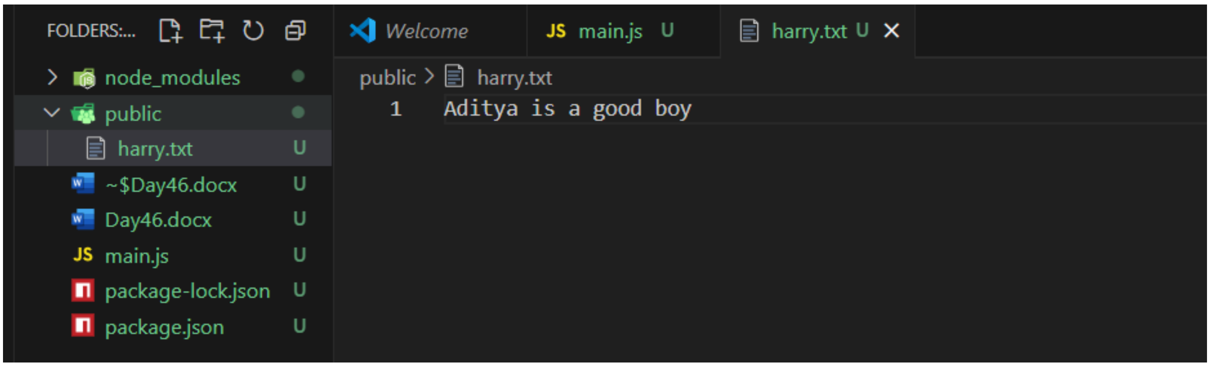
Why Middleware is Useful

- Logging requests
- Authenticating users
- Parsing request bodies (JSON, URL-encoded)
- Handling errors
- Serving static files

Without middleware, you’d have to manually write the same logic inside every route — which is messy and repetitive.

Now, we are using the built in middleware of express js. For which we will first create a “public” folder and inside it we will create “harry.txt” and some content in it.

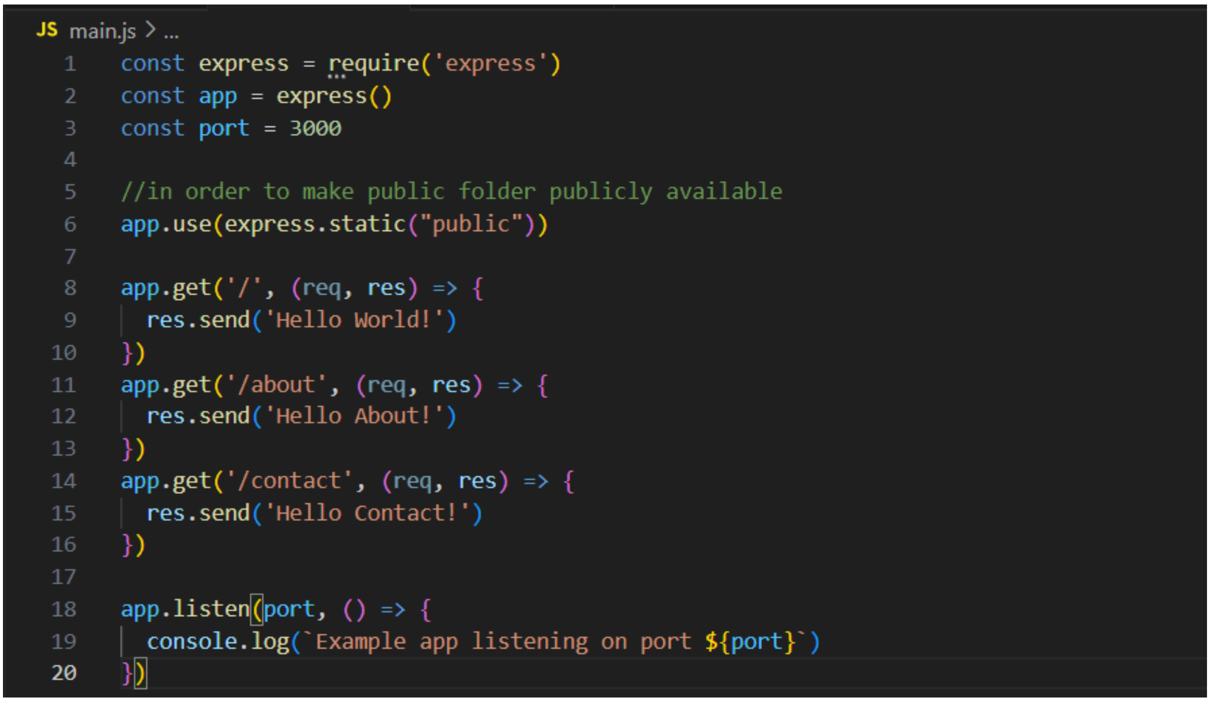
Public/harry.txt:



```
Aditya is a good boy
```

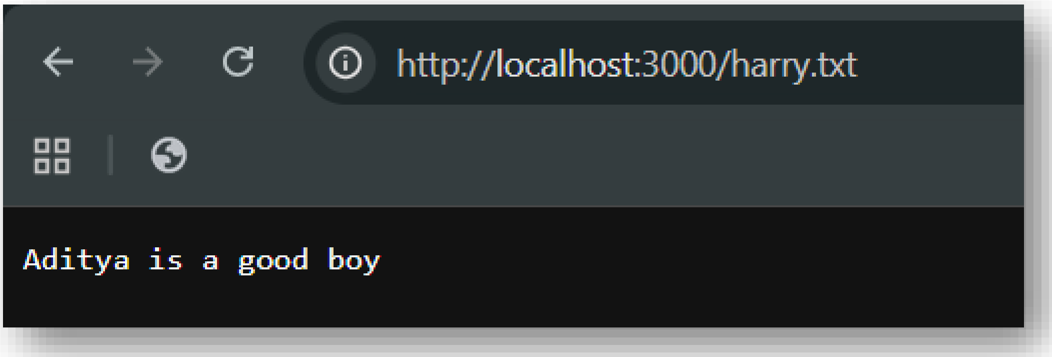
Now, we will be allowing our browser to use it using the following code:

Main.js:



```
const express = require('express')
const app = express()
const port = 3000

//in order to make public folder publicly available
app.use(express.static("public"))

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

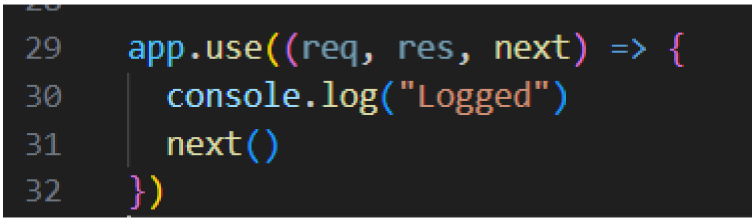
Output: at “/harry.txt”



```
Aditya is a good boy
```

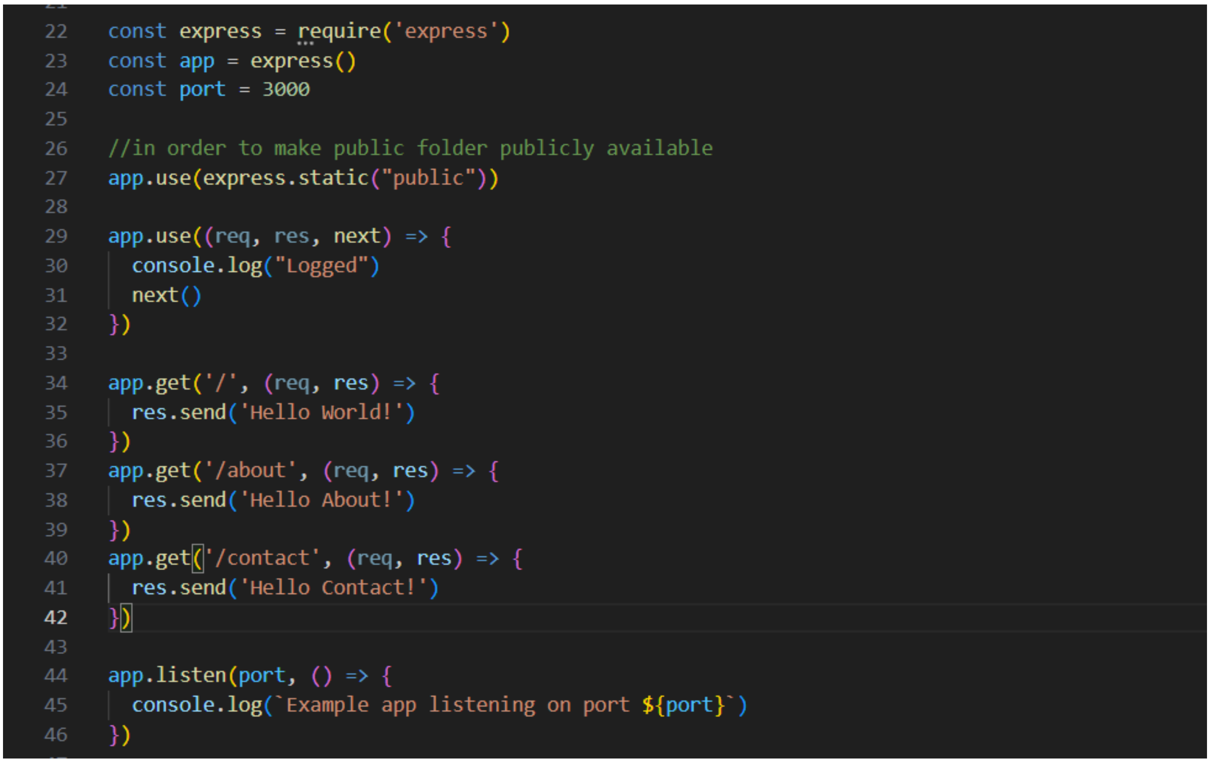
What basically, happened in this example is that first the request is made, then the request Is checked if it is in public folder or not, since it is in public folder. It sends the data to the response form.

Now, we can write our own middleware using the following part of code:



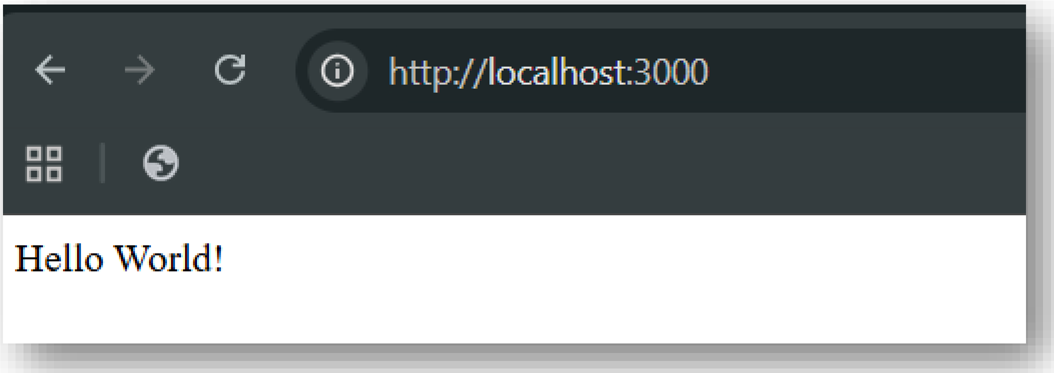
```
app.use((req, res, next) => {
  console.log("Logged")
  next()
})
```

We can see the whole code as:



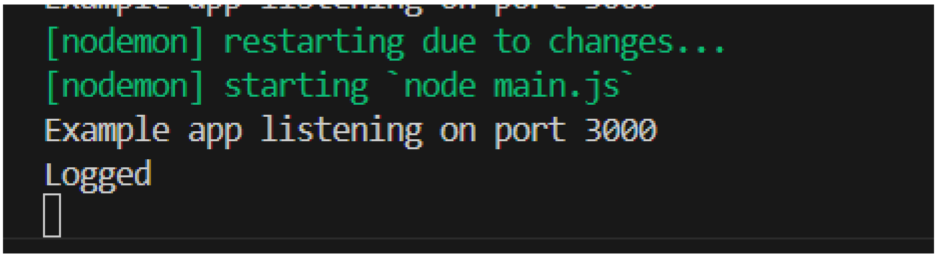
```
const express = require('express')
const app = express()
const port = 3000

//in order to make public folder publicly available
app.use(express.static("public"))

app.use((req, res, next) => {
  console.log("Logged")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Output: browser



Hello World!

Terminal:

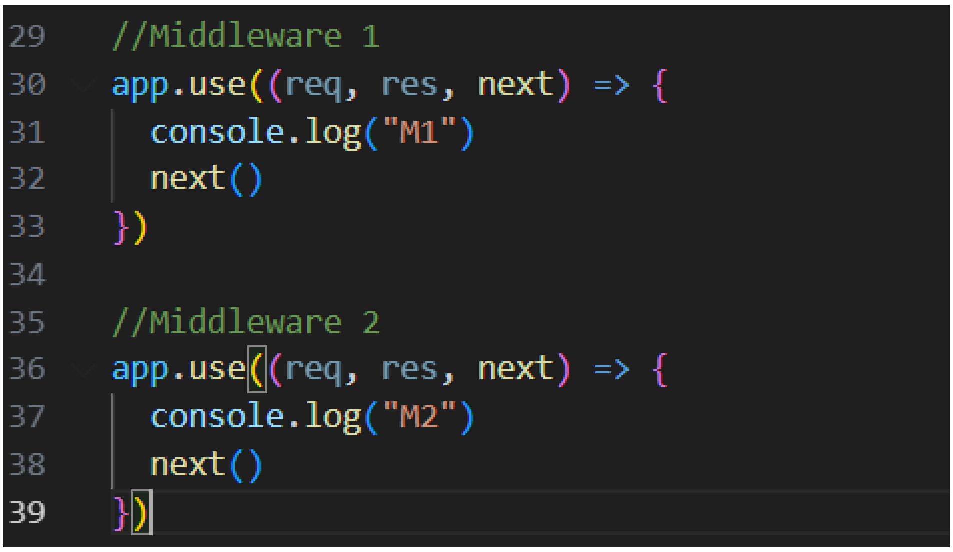


```
Example app listening on port 3000
[nodemon] restarting due to changes...
[nodemon] starting `node main.js`
Example app listening on port 3000
Logged
```

So, what basically happened? We used the middleware, and as since it has the parameters, req it deals with the request, res for response, and next to say that move to the next middleware. So, when website was loaded at main page, then the "hello world!" got printed while "Logged" is observed on the terminal.

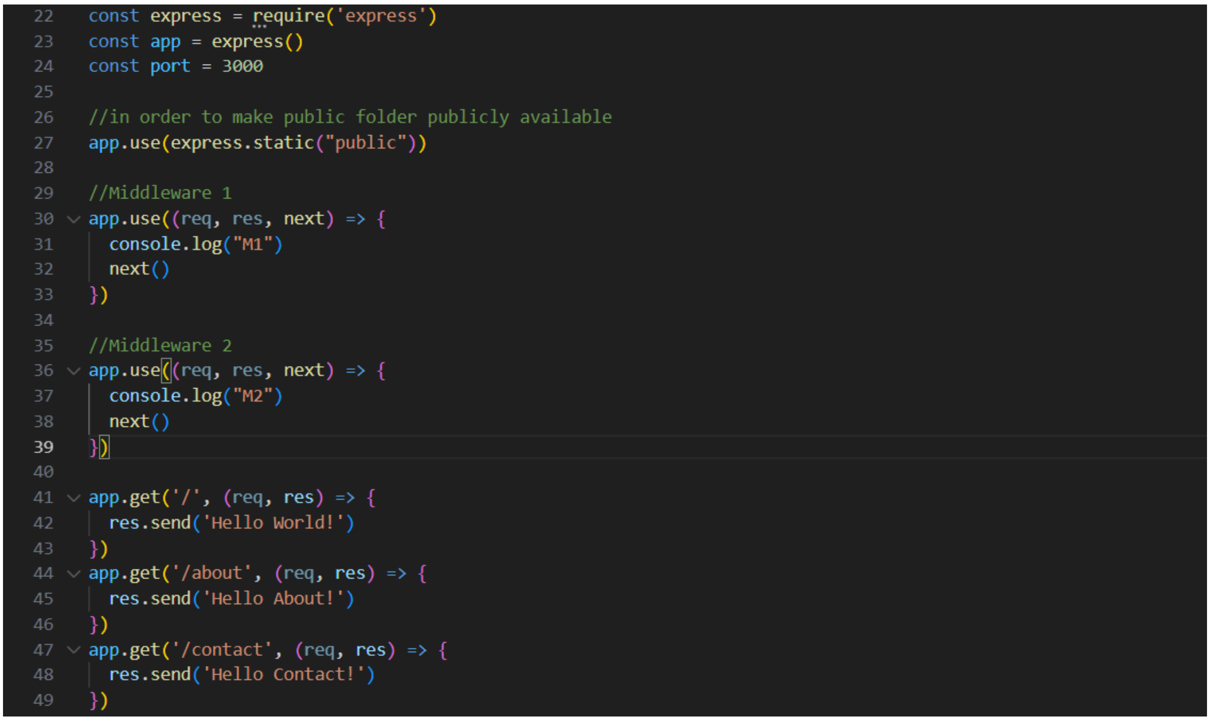
Now, to better understand the next() we will add one more middleware as shown below:

Snippet:



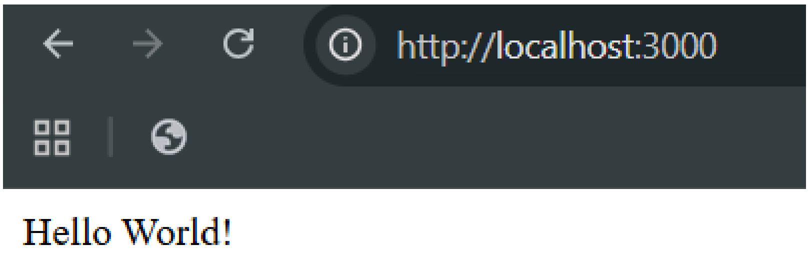
```
//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  next()
})

//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})
```

Whole code:



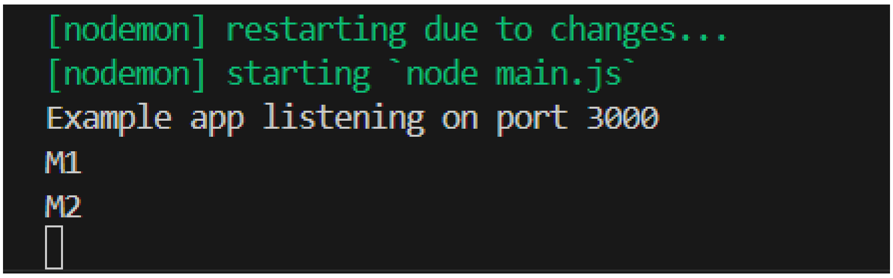
```
const express = require('express')
const app = express()
const port = 3000

//in order to make public folder publicly available
app.use(express.static("public"))

//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  next()
})

//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})
```

Output: browser



Hello World!

Output: terminal



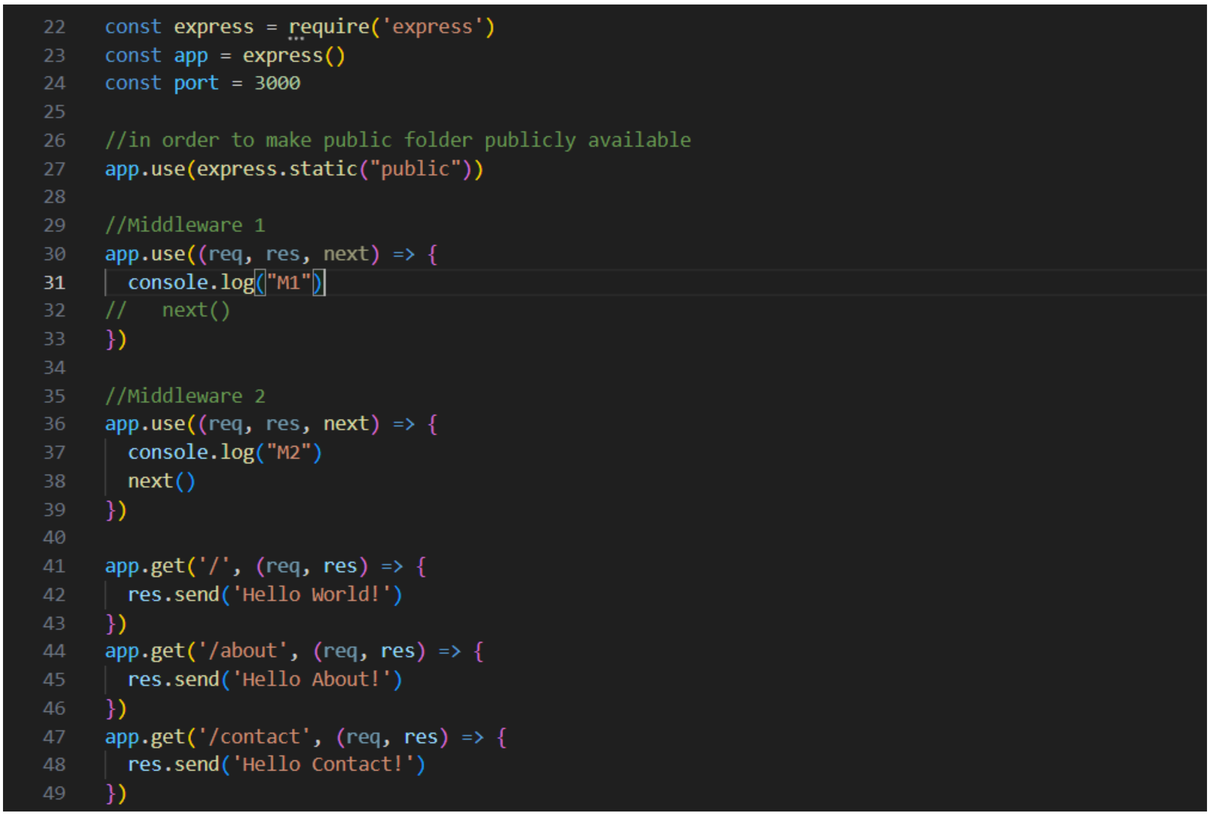
```
[nodemon] restarting due to changes...
[nodemon] starting `node main.js`
Example app listening on port 3000
M1
M2
```

Clearly, first middleware 1 ran, then printed M1 and then due to next() it says to go to the next middleware, and that’s why M2 printed.

What if we remove the “next()”?

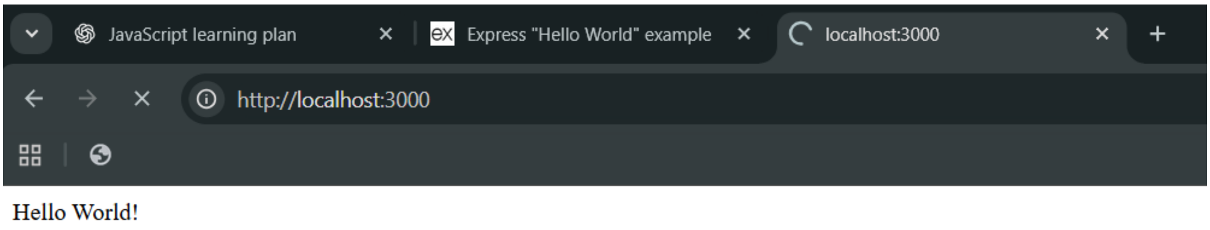
Then we will observe that the response will not come, although the terminal will show the M1 but in browser it will remain loading.

Main.js:



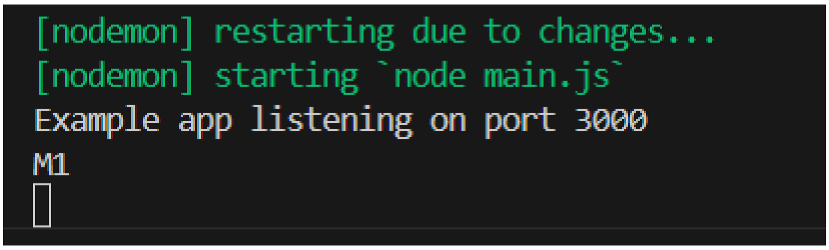
```
const express = require('express')
const app = express()
const port = 3000

//in order to make public folder publicly available
app.use(express.static("public"))

//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
//   next()
})

//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})
```

Output: see it is loading.



Hello World!

Terminal:

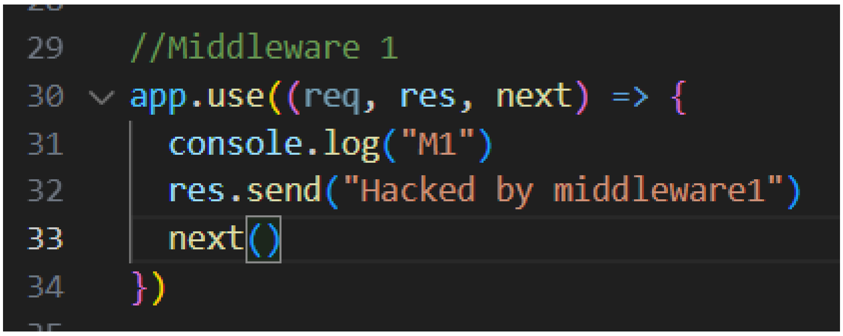


```
[nodemon] restarting due to changes...
[nodemon] starting `node main.js`
Example app listening on port 3000
M1
```

So, basically the control never went from middleware 1 as we had removed “next()”.

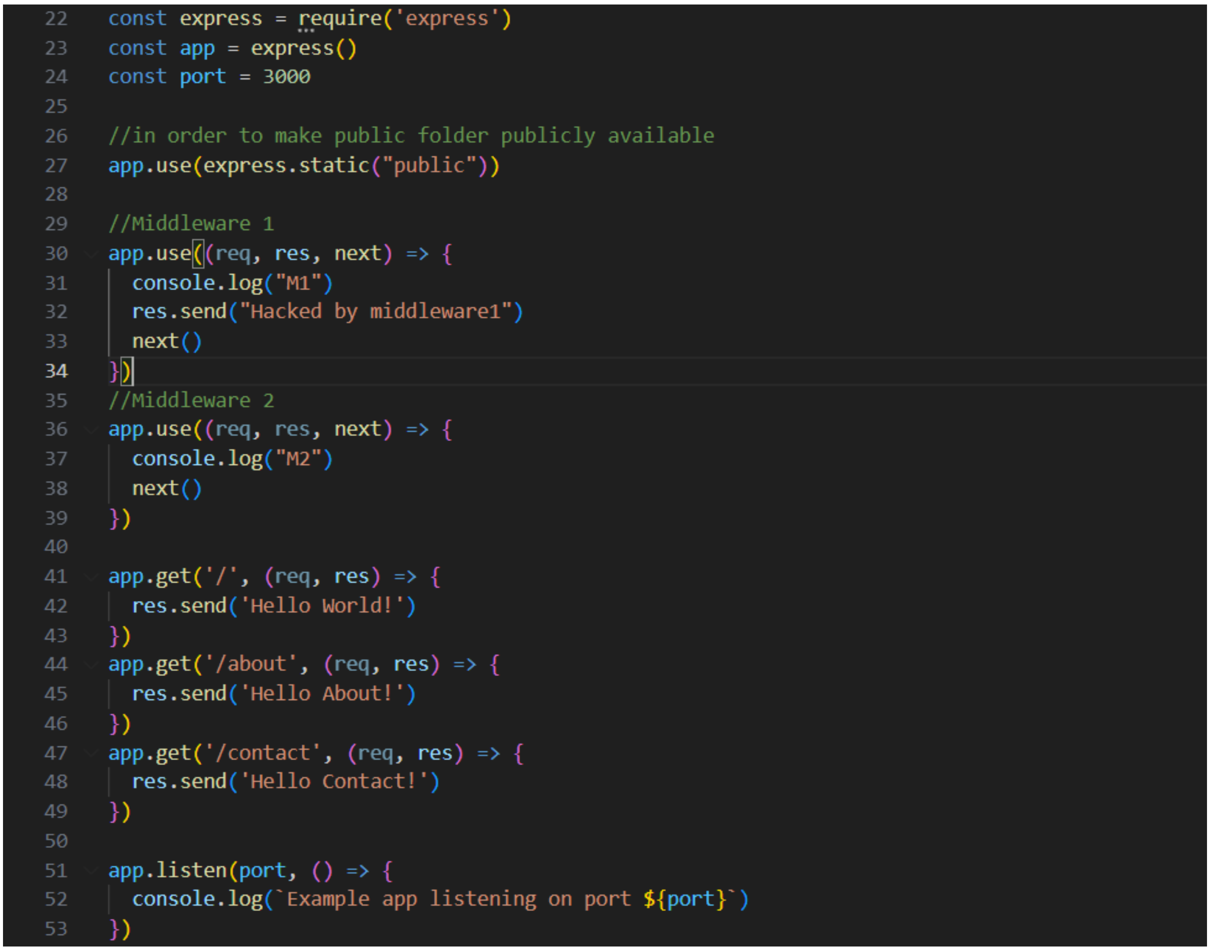
Now, suppose if request is sent and still you are giving the controls then it will load the output at browser but at terminal it will show error.

Snippet:



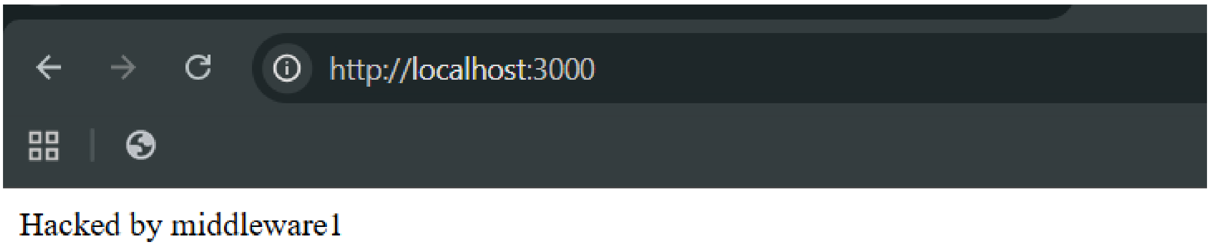
```
//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  res.send("Hacked by middleware1")
  next()
})
```

Code:



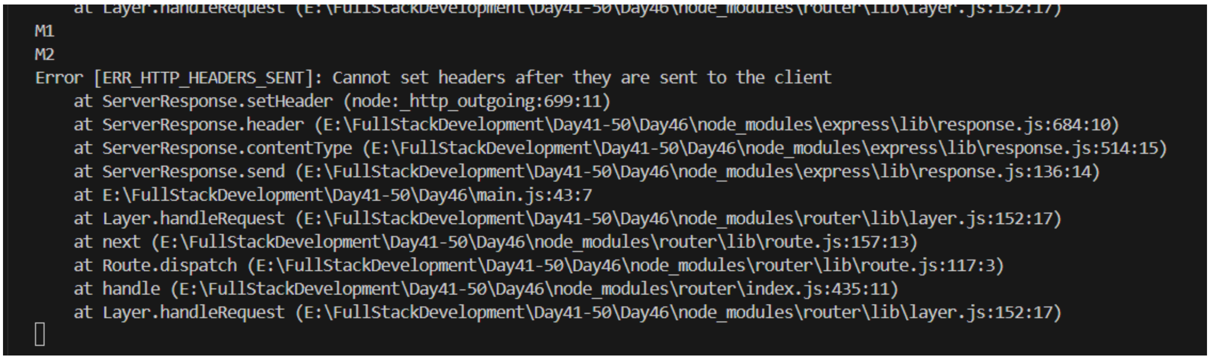
```
const express = require('express')
const app = express()
const port = 3000

//in order to make public folder publicly available
app.use(express.static("public"))

//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  res.send("Hacked by middleware1")
  next()
})
//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Output: at browser



Hacked by middleware1

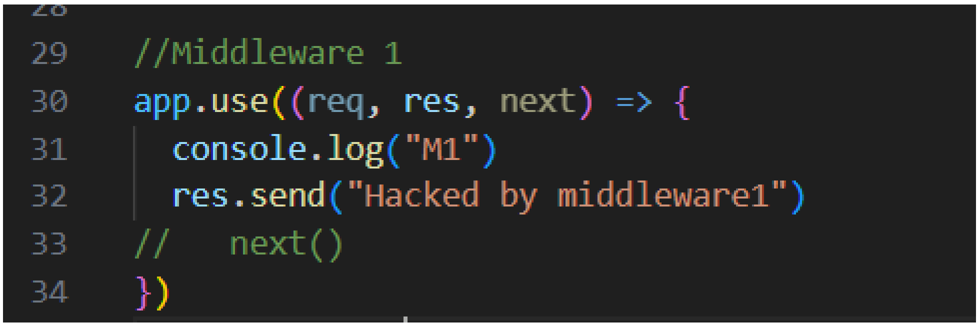
At terminal: error



```
    at Layer.handleRequest (E:\FullStackDevelopment\Day41-50\Day46\node_modules\router\lib\layer.js:152:17)
M1
M2
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
    at ServerResponse.setHeader (node:_http_outgoing:699:11)
    at ServerResponse.header (E:\FullStackDevelopment\Day41-50\Day46\node_modules\express\lib\response.js:684:10)
    at ServerResponse.contentType (E:\FullStackDevelopment\Day41-50\Day46\node_modules\express\lib\response.js:514:15)
    at ServerResponse.send (E:\FullStackDevelopment\Day41-50\Day46\node_modules\express\lib\response.js:136:14)
    at E:\FullStackDevelopment\Day41-50\Day46\main.js:43:7
    at Layer.handleRequest (E:\FullStackDevelopment\Day41-50\Day46\node_modules\router\lib\layer.js:152:17)
    at next (E:\FullStackDevelopment\Day41-50\Day46\node_modules\router\lib\route.js:157:13)
    at Route.dispatch (E:\FullStackDevelopment\Day41-50\Day46\node_modules\router\lib\route.js:117:3)
    at handle (E:\FullStackDevelopment\Day41-50\Day46\node_modules\router\index.js:435:11)
    at Layer.handleRequest (E:\FullStackDevelopment\Day41-50\Day46\node_modules\router\lib\layer.js:152:17)
```

But what if we want that response is sent by the middleware 1 and the error doesn't occur, for this we will remove the next() from the middleware 1. (rest code remains same)

Snippet:

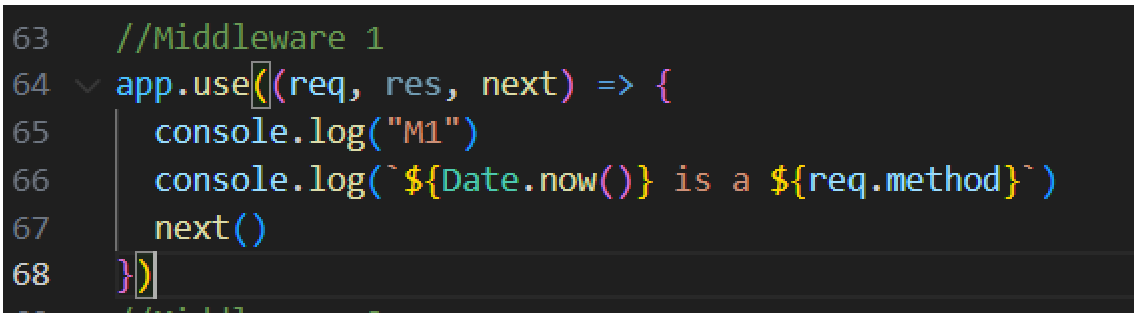


```
//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  res.send("Hacked by middleware1")
//    next()
})
```

"So, basically till now, we saw that middleware can change the request, or can also send the request and also can give control to the next middleware."

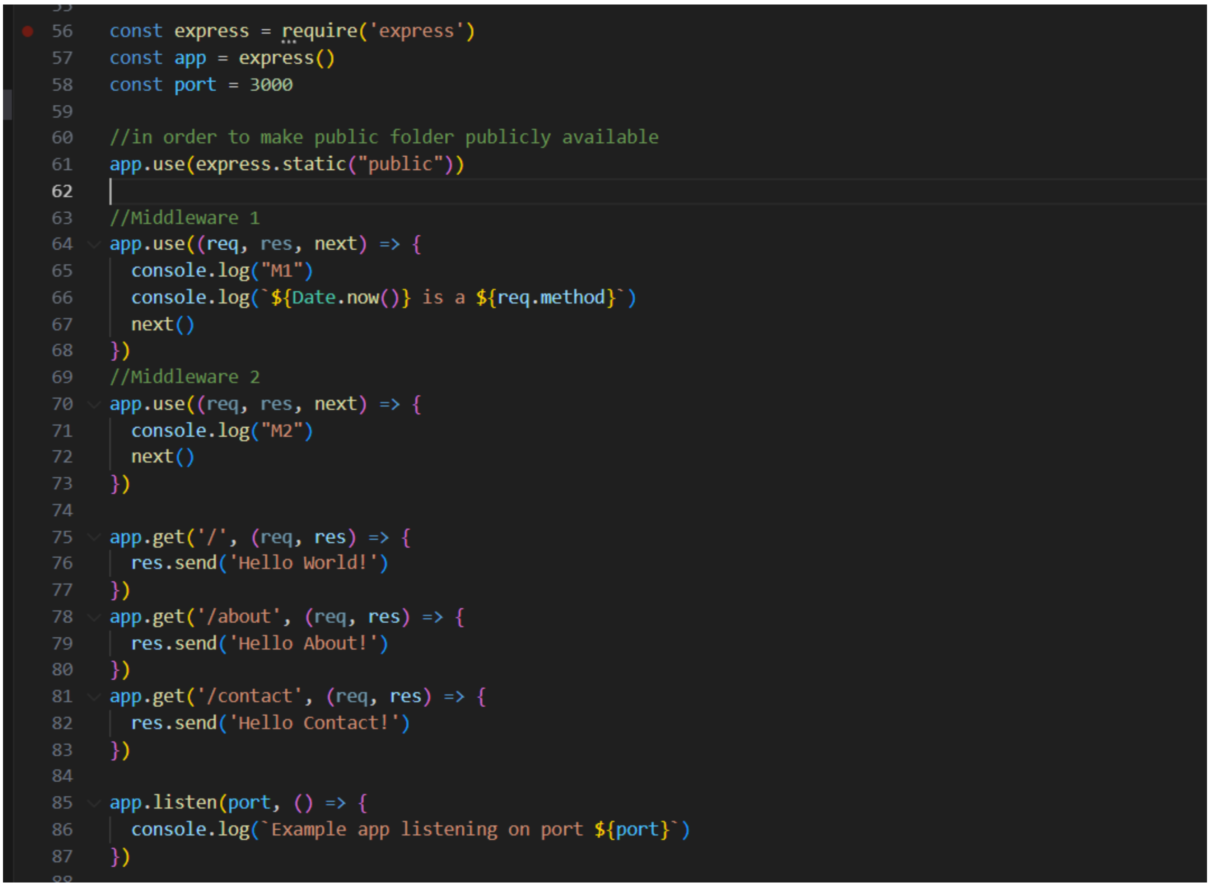
So, why we need middleware? In below example we will understand that we need middleware in order to log the data.

Snippet: we are basically logging the date and request method.



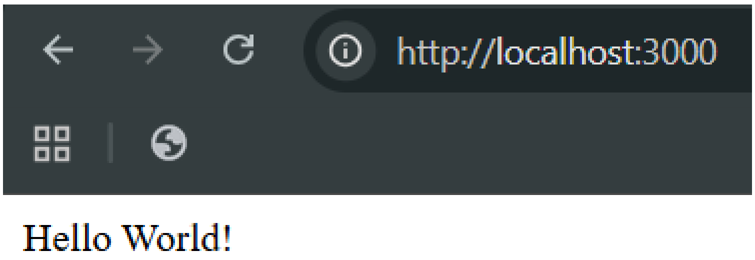
```
//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  console.log(`${Date.now()} is a ${req.method}`)
  next()
})
```

Code:



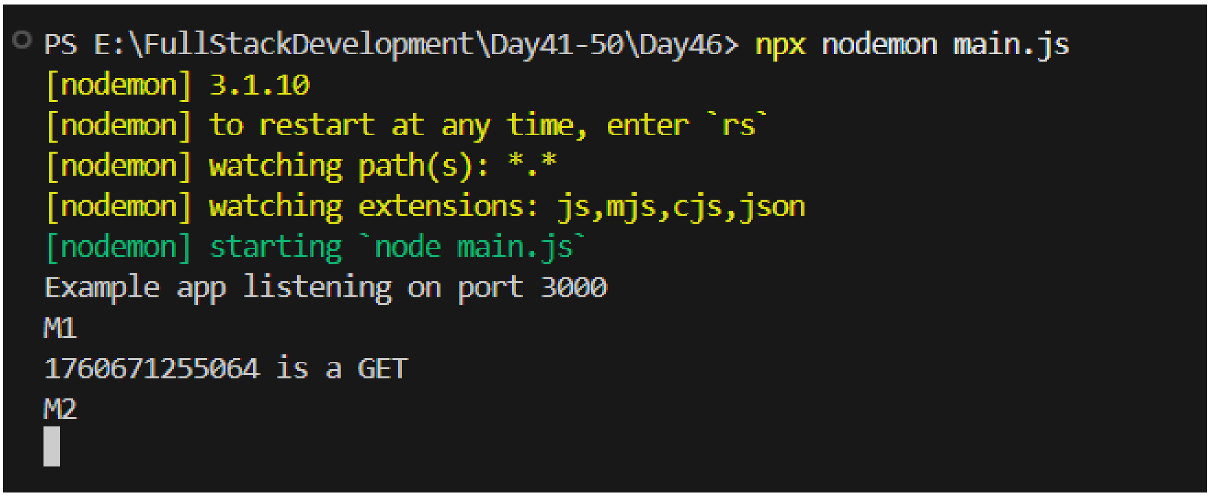
```
const express = require('express')
const app = express()
const port = 3000

//in order to make public folder publicly available
app.use(express.static("public"))

//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  console.log(`${Date.now()} is a ${req.method}`)
  next()
})
//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Output: browser



Hello World!

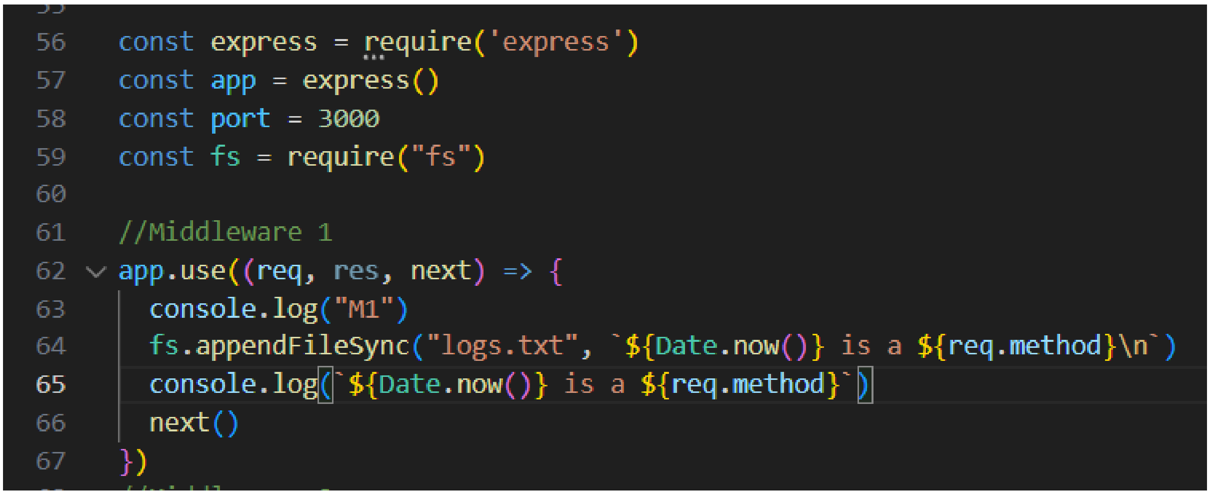
Output: terminal



```
PS E:\FullStackDevelopment\Day41-50\Day46> npx nodemon main.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node main.js`
Example app listening on port 3000
M1
1760671255064 is a GET
M2
```

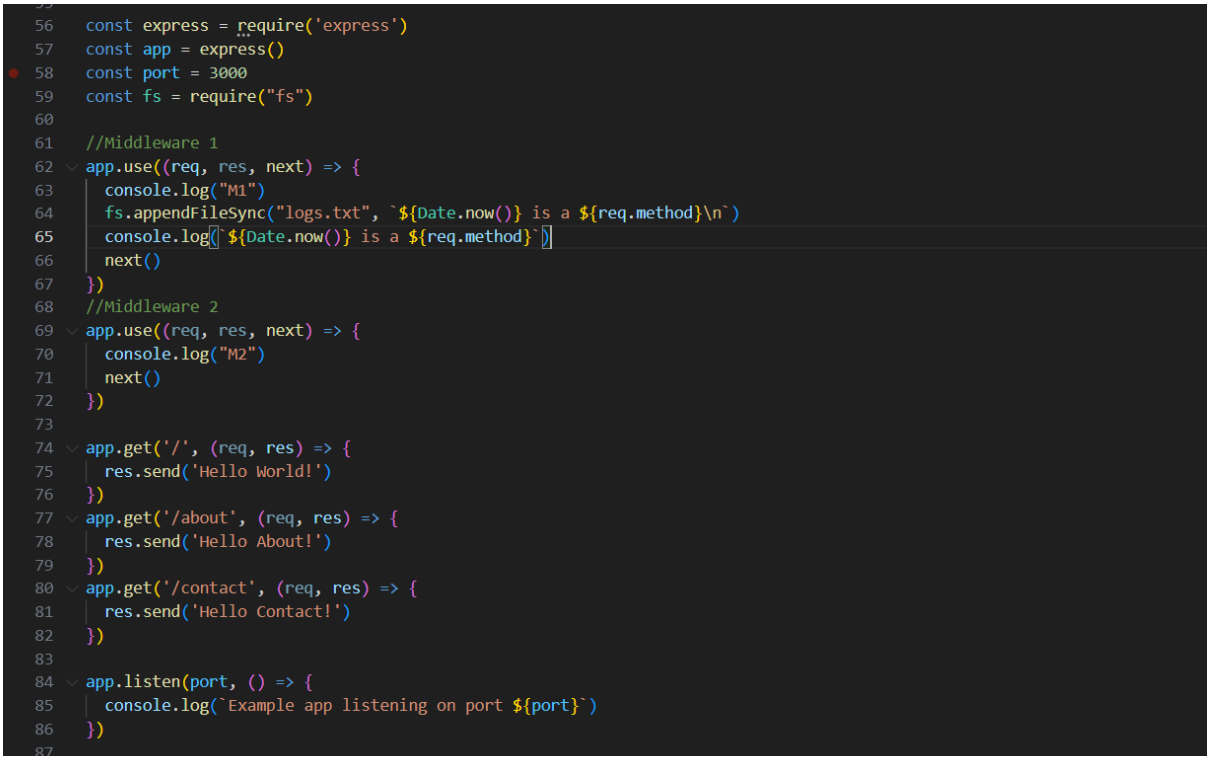
Now, we can also write and append this logged data in a text file: for which we will use “fs” module.

Snippet:



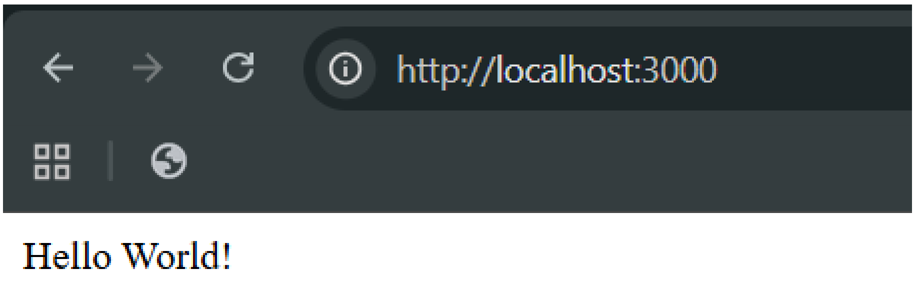
```
const express = require('express')
const app = express()
const port = 3000
const fs = require("fs")

//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  fs.appendFileSync("logs.txt", `${Date.now()} is a ${req.method}\n`)
  console.log(`${Date.now()} is a ${req.method}`)
  next()
})
```

Code:



```
const express = require('express')
const app = express()
const port = 3000
const fs = require("fs")

//Middleware 1
app.use((req, res, next) => {
  console.log("M1")
  fs.appendFileSync("logs.txt", `${Date.now()} is a ${req.method}\n`)
  console.log(`${Date.now()} is a ${req.method}`)
  next()
})
//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

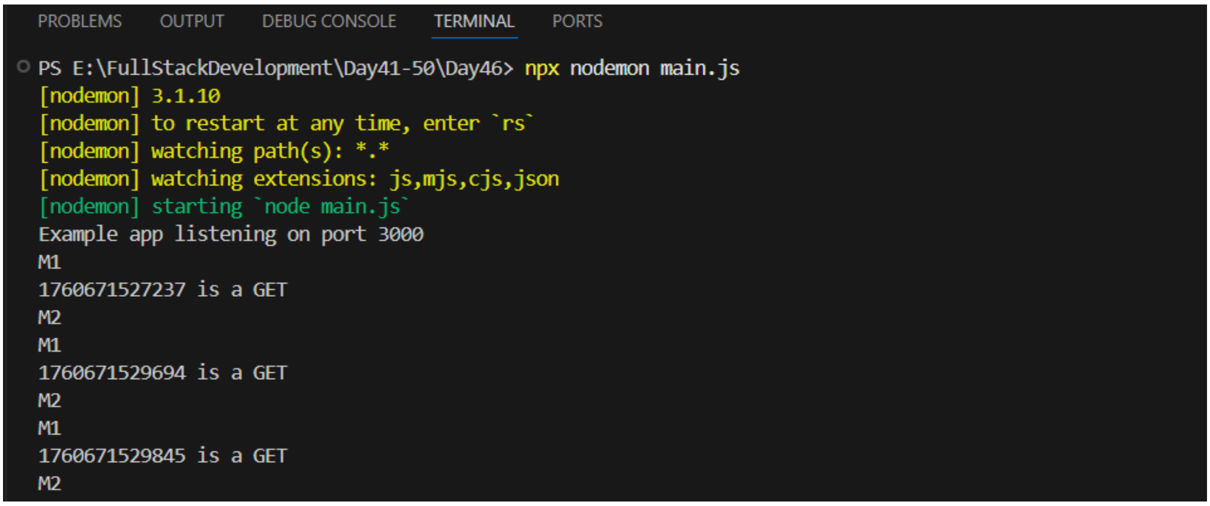
Output: browser



http://localhost:3000

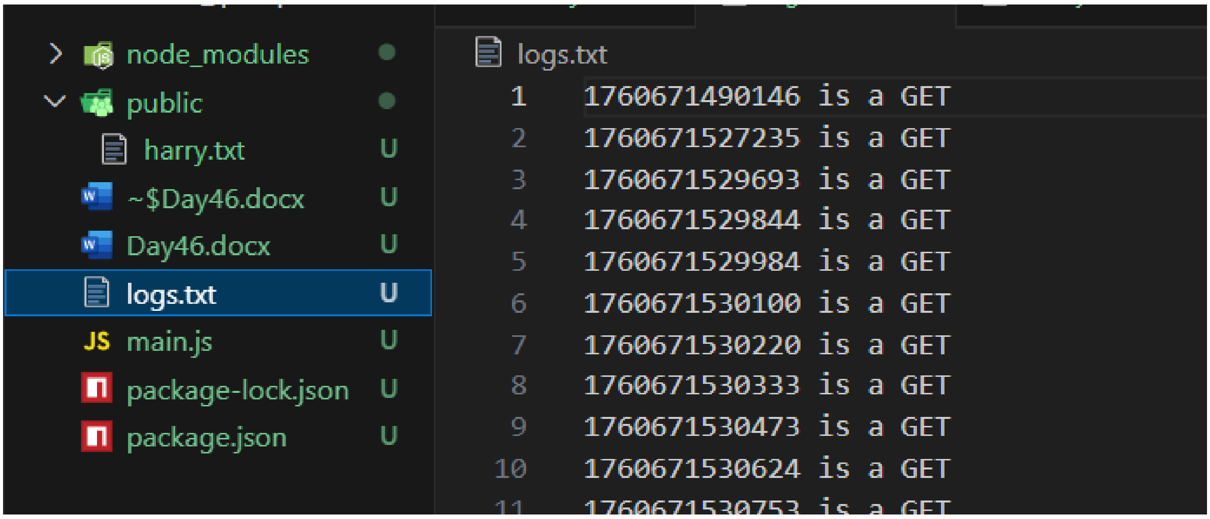
Hello World!

Output: terminal



```
PS E:\FullStackDevelopment\Day41-50\Day46> npx nodemon main.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node main.js`
Example app listening on port 3000
M1
1760671527237 is a GET
M2
M1
1760671529694 is a GET
M2
M1
1760671529845 is a GET
M2
```

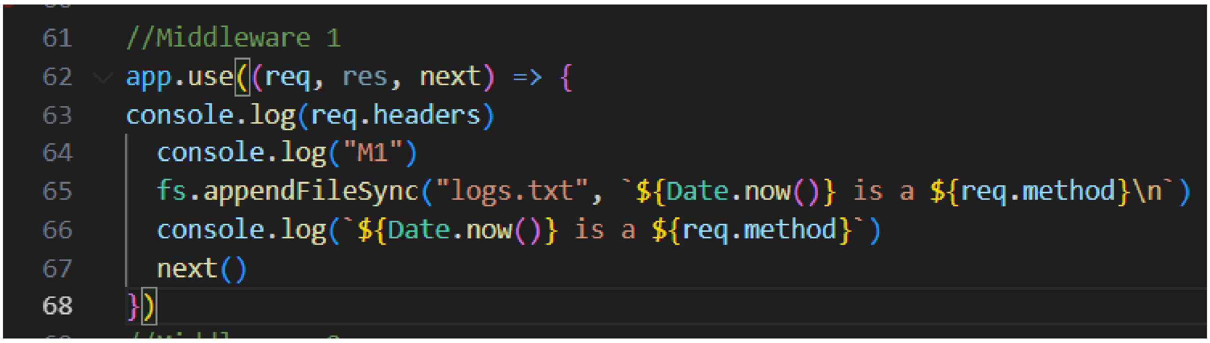
Output: logs.txt



```
1760671490146 is a GET
1760671527235 is a GET
1760671529693 is a GET
1760671529844 is a GET
1760671529984 is a GET
1760671530100 is a GET
1760671530220 is a GET
1760671530333 is a GET
1760671530473 is a GET
1760671530624 is a GET
1760671530753 is a GET
```

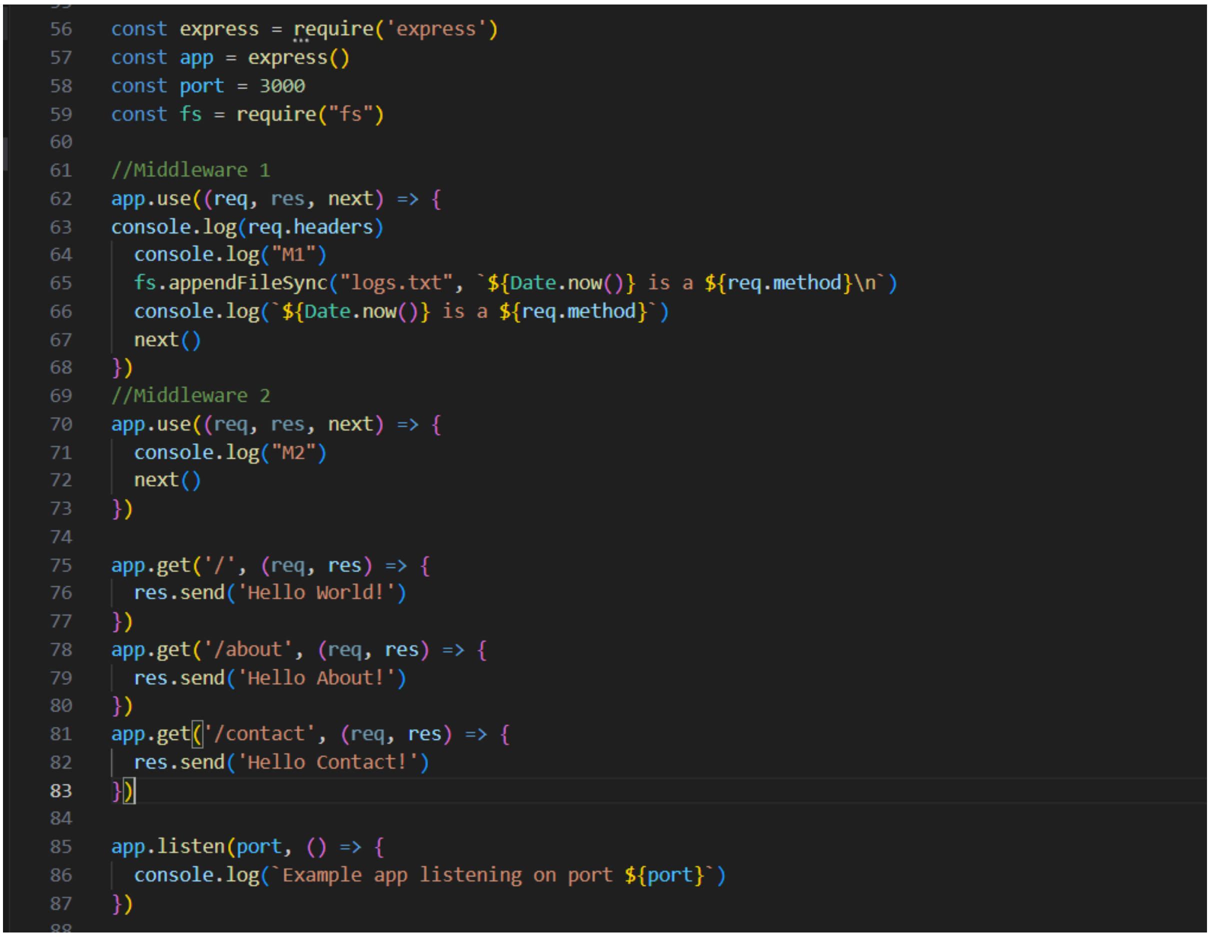
Now, we can also get the headers:

Snippet: we used req.headers and logged it



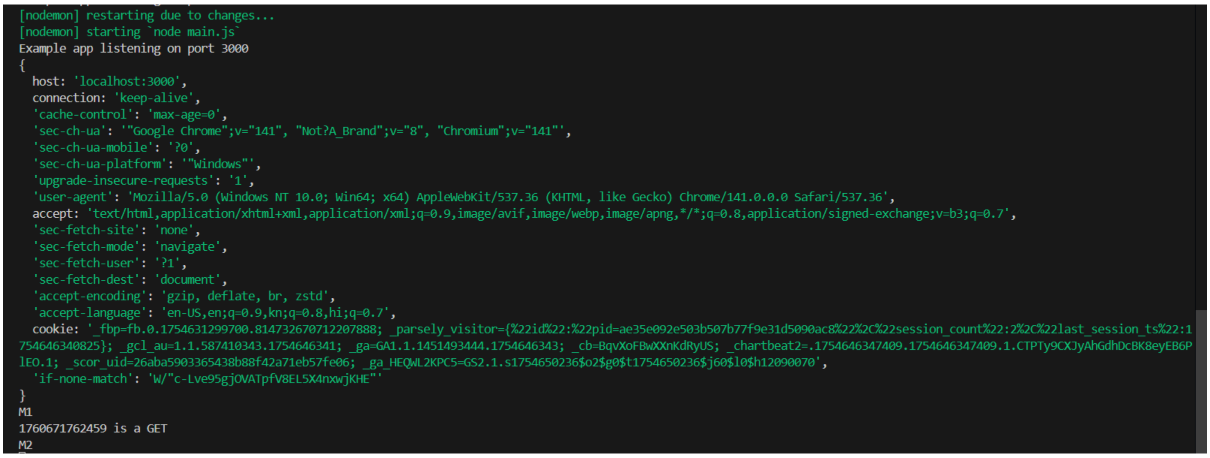
```
//Middleware 1
app.use((req, res, next) => {
console.log(req.headers)
    console.log("M1")
    fs.appendFileSync("logs.txt", `${Date.now()} is a ${req.method}\n`)
    console.log(`${Date.now()} is a ${req.method}`)
    next()
})
```

Code:



```
const express = require('express')
const app = express()
const port = 3000
const fs = require("fs")

//Middleware 1
app.use((req, res, next) => {
console.log(req.headers)
  console.log("M1")
  fs.appendFileSync("logs.txt", `${Date.now()} is a ${req.method}\n`)
  console.log(`${Date.now()} is a ${req.method}`)
  next()
})
//Middleware 2
app.use((req, res, next) => {
  console.log("M2")
  next()
})

app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.get('/about', (req, res) => {
  res.send('Hello About!')
})
app.get('/contact', (req, res) => {
  res.send('Hello Contact!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Output: terminal



```
[nodemon] restarting due to changes...
[nodemon] starting `node main.js`
Example app listening on port 3000
{
  host: 'localhost:3000',
  connection: 'keep-alive',
  'cache-control': 'max-age=0',
  'sec-ch-ua': '"Google Chrome";v="141", "Not?A_Brand";v="8", "Chromium";v="141"',
  'sec-ch-ua-mobile': '?0',
  'sec-ch-ua-platform': '"Windows"',
  'upgrade-insecure-requests': '1',
  'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36',
  accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7',
  'sec-fetch-site': 'none',
  'sec-fetch-mode': 'navigate',
  'sec-fetch-user': '?1',
  'sec-fetch-dest': 'document',
  'accept-encoding': 'gzip, deflate, br, zstd',
  'accept-language': 'en-US,en;q=0.9,kn;q=0.8,hi;q=0.7',
  cookie: '_fbp=fb.0.1754631299700.814732670712207888; _parsely_visitor={%22id%22:%22pid=ae35e092e503b507b77f9e31d5090ac8%22%2C%22session_count%22:2%2C%22last_session_ts%22:1
754646340825}; _gcl_au=1.1.587410343.1754646341; _ga=GA1.1.1451493444.1754646343; _cb=BqvXoFBwXXnKdRyUS; _chartbeat2=.1754646347409.1754646347409.1.CTPTy9CXJyAhGdhDcBK8eyEB6P
lEO.1; _scor_uid=26aba5903365438b88f42a71eb57fe06; _ga_HEQWL2KPC5=GS2.1.s1754650236$o2$g0$t1754650236$j60$l0$h12090070',
  'if-none-match': 'W/"c-Lve95gjOVATpfV8EL5X4nxwjKHE"'
}
M1
1760671762459 is a GET
M2
```

Types of Middleware

Type	**Description**	**Example**
Application-level	Used for all routes or specific routes	app.use()
Router-level	Used for routes in a router	router.use()
Third-party	Middleware from npm packages	body-parser, cors, helmet
Error-handling	Handles errors in routes	(err, req, res, next) => {...}
Built-in	Middleware provided by Express	express.json(), express.static()

--The End--