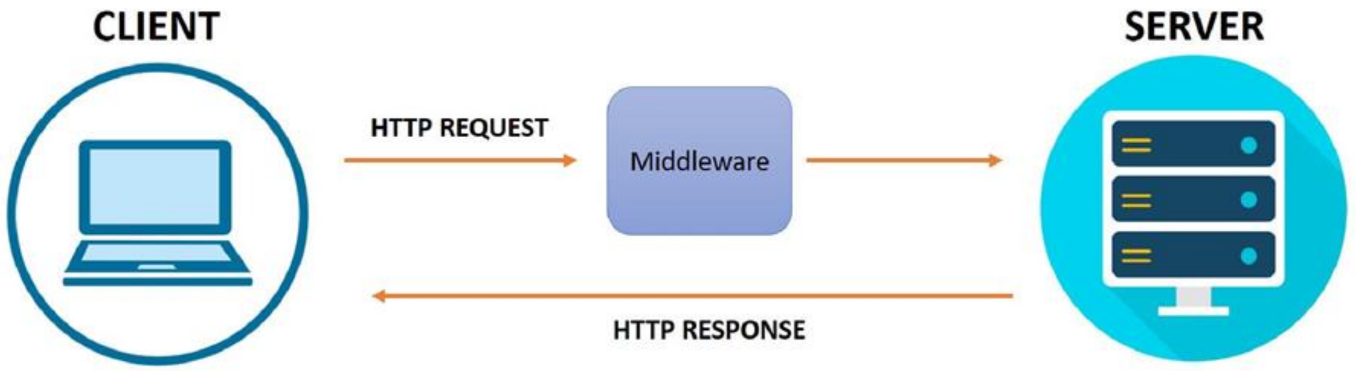
Collection of JavaScript based technologies used to develop web applications.

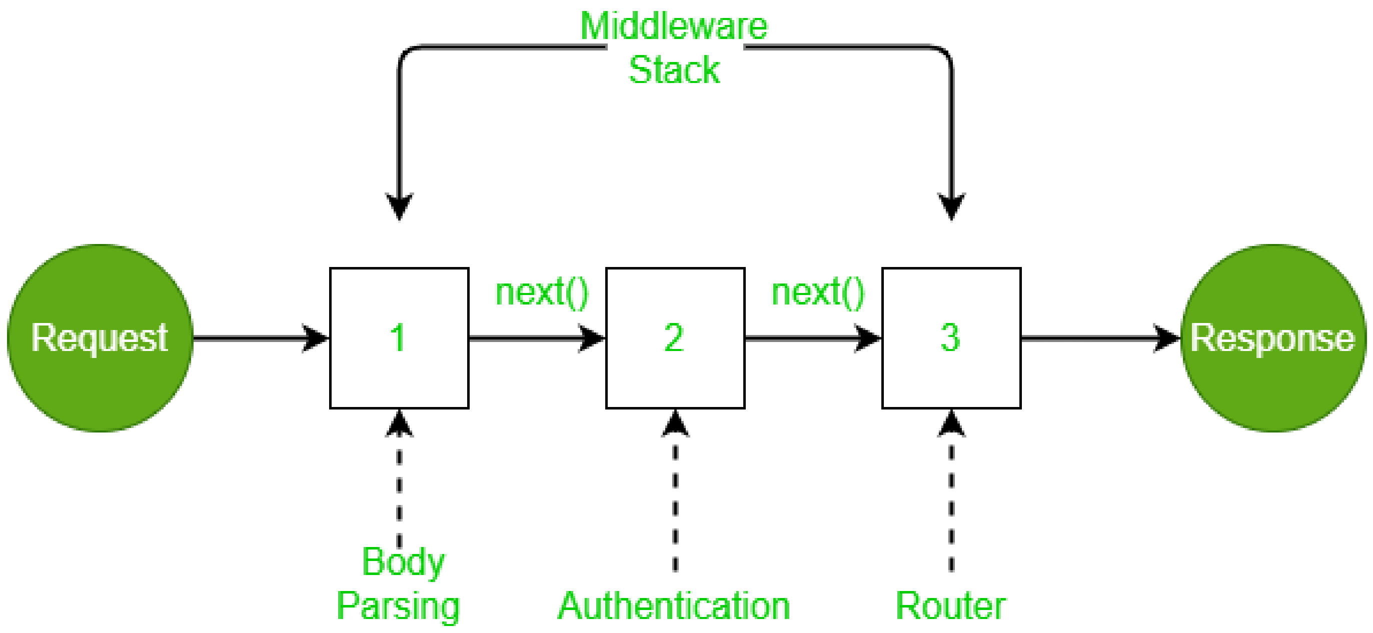
node JS™

express

Laiba Imran

Middleware for Express





Middleware for Express

A software with functions that have access to: **Request Object** | **Response Object**

Executes during the request and the response cycle

Can be used for:

Logger
- Logs user information

Authentication
- Protects the routes

Parsing JSON Data
- Executes any code
- Makes changes to request and response objects
- Ends the request-response cycle
- Calls the next middleware in the stack

Express middleware includes application-level, router-level, and error handling functionalities.

It can be built in or extracted from a third-party module.

Middleware for Express

- Middleware is a specific part of the backend code that processes incoming requests before they reach the main logic (e.g., route handlers) and after responses are generated but before they are sent back to the client.
- Middleware serves as a layer between the client request and the core backend logic.
- Usually focused on pre-processing or post-processing requests

Middleware for Express

- Execute any code: Perform operations like logging, authentication, etc.
- Make changes to request and response objects: Modify the req or res before the final handler is executed.
- End the request-response cycle: Some middleware can send the response to the client directly, stopping further execution.
- Call the next middleware in the stack: Middleware can decide if it should pass control to the next middleware using next().

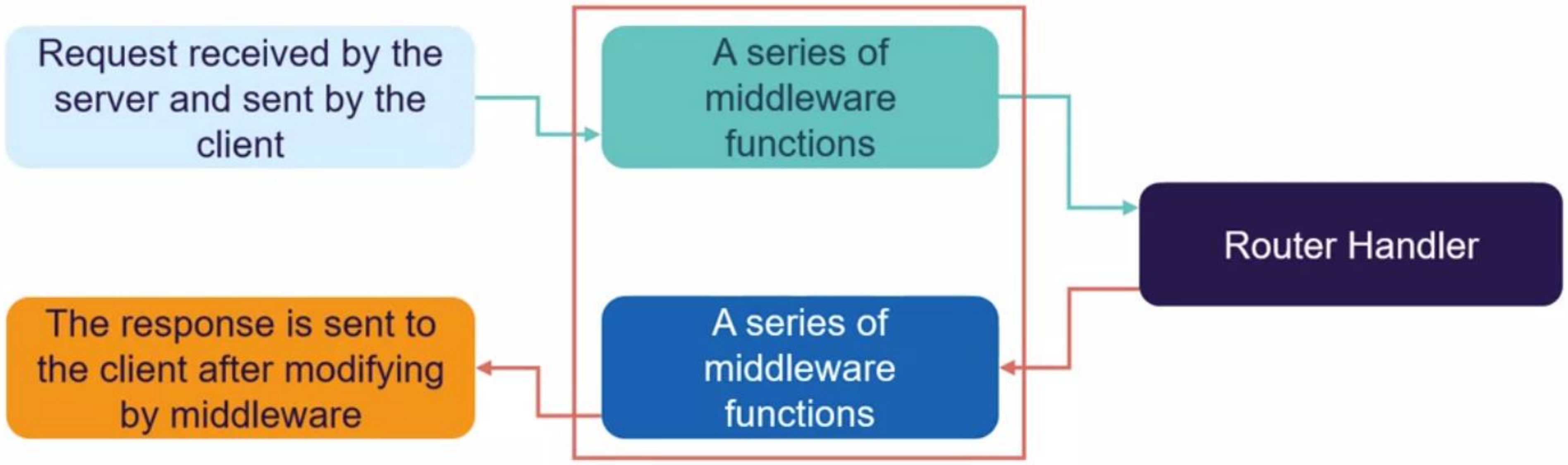
Middleware for Express

- Middleware can be used for:
  - **Logger**: Logs user information, requests, and responses.
  - **Authentication**: Protects routes by checking user authorization (e.g., using JWT tokens).
  - **Parsing JSON Data**: Automatically processes incoming JSON request bodies (e.g., express.json()).

Middleware for Express

- Middleware in Express can be:
  - Application-level middleware: Bound to the entire application using app.use() or app.METHOD() and executes for all routes.
  - Router-level middleware: Associated with specific routes using router.use() or router.METHOD() and executes for routes defined within that router.
  - Error-handling middleware: Handles errors during the request-response cycle. Defined with four parameters (err, req, res, next).
  - Built-in middleware: Provided by Express (e.g., express.static, express.json, etc.).
  - Third-party middleware: Developed by external packages (e.g., body-parser, morgan, etc.)

Middleware for Express

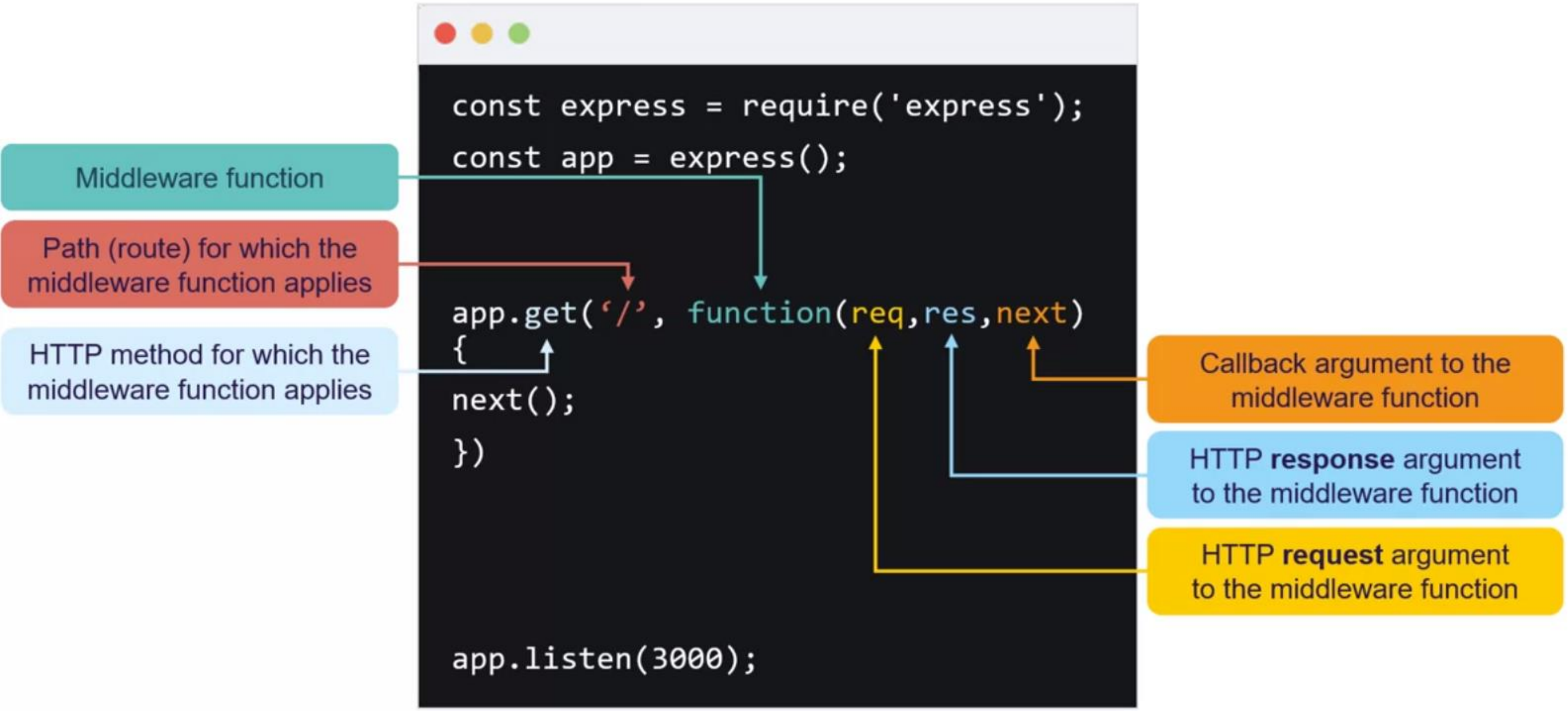

Request received by the server and sent by the client
A series of middleware functions
Router Handler
A series of middleware functions
The response is sent to the client after modifying by middleware

next() is a callback function that passes control to the next middleware function.

The chain ends if the next() method in the series of middleware is not called.

The request will be left hanging if the request-response cycle does not end.

Middleware for Express

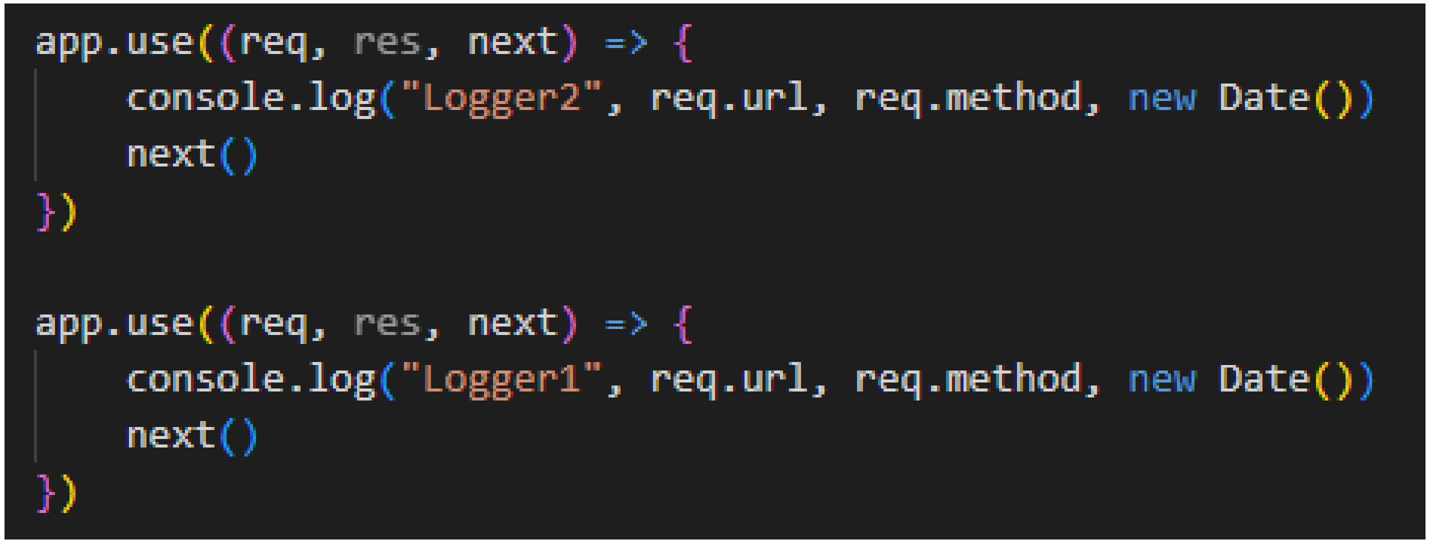


```
const express = require('express');
const app = express();


app.get('/', function(req,res,next)
{
next();
})



app.listen(3000);
```

Middleware function
Path (route) for which the middleware function applies
HTTP method for which the middleware function applies
Callback argument to the middleware function
HTTP **response** argument to the middleware function
HTTP **request** argument to the middleware function

Global Middleware

- Executes in an order
- Executes on every request



```
app.use((req, res, next) => {
    console.log("Logger2", req.url, req.method, new Date())
    next()
})

app.use((req, res, next) => {
    console.log("Logger1", req.url, req.method, new Date())
    next()
})
```

Global Middleware

```
const LoggerMiddleware = (req,res,next) =>{
console.log(`Logged ${req.url} ${req.method}
-- ${new Date()}`)
    next();
}


app.use(LoggerMiddleware)
```

Middleware Function—Logger

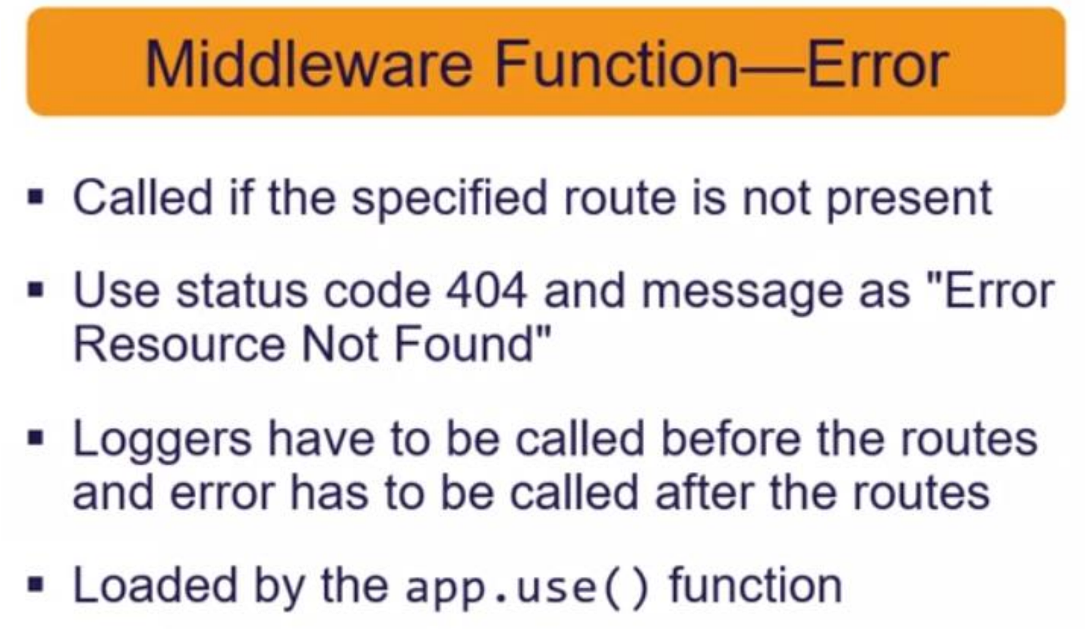
- Helps trace the errors of the application
- Helps in creating custom loggers
- Takes three parameters:
  - request
  - response
  - next()
- Requires the `app.use()` function to load

```
Logged  /  GET --
Mon Nov 29, 2021 19:10:53 GMT+0530
(India Standard Time)
```

National University
Of Computer and Emerging Sciences

```
const express = require('express');
const app = express();
const routes = require('./routes'); // Import routes from routes.js

// Global Middleware
app.use(express.json()); // Parses JSON request bodies
const LoggerMiddleware = (req, res, next) => {
   console.log(`${req.method} ${req.url}`); // Logs the request method and URL
   next(); // Passes control to the next middleware or route handler
});

//use the global middleware
app.use(LoggerMiddleware)
// Use routes defined in routes.js
app.use('/api', routes); // The '/api' path will use the routes from routes.js

app.listen(3000, () => {
   console.log('Server is running on port 3000');
});
```

Global Middleware



Middleware Function—Error

- Called if the specified route is not present
- Use status code 404 and message as "Error Resource Not Found"
- Loggers have to be called before the routes and error has to be called after the routes
- Loaded by the `app.use()` function

Error Handling Middleware

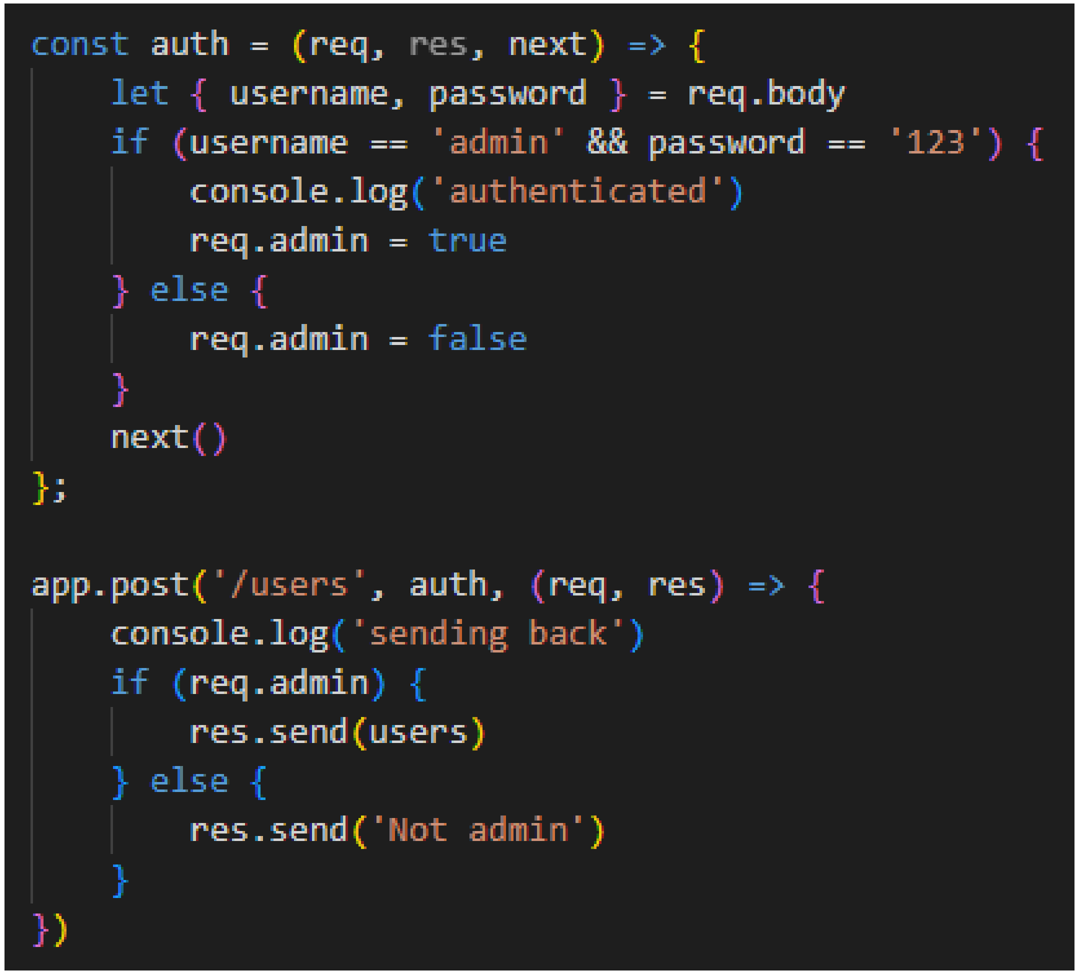
- Error handling middleware has an extra argument `err`, e.g. `(err, req, res, next)`
- Calling `next(err)` will bypass the rest of the regular middleware and pass control to the next error handling middleware
  - `err` is an Error object
- Express adds a default error handling middleware at the end of the middleware chain

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

// First route that may generate an error
app.get('/route1', (req, res, next) => {
   // Simulate an error
   const error = new Error('Error from Route 1');
   next(error); // Pass the error to the error middleware
});

// Error-handling middleware
app.use((err, req, res, next) => {
   console.error(err.message); // Log the error message
   res.status(500).send(`An error occurred: ${err.message}`); // Send error response
});

app.listen(3000, () => {
   console.log('Server is running on port 3000');
});
```

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

// First route that may generate an error
app.get('/route1', (req, res, next) => {
  // Simulate an error
  const error = new Error('Error from Route 1');
  next(error); // Pass the error to the error middleware
});

app.get('/route2', (req, res) => {
  res.send('This is Route 2'); // This will not be executed if Route 1 has an error
});

// Error-handling middleware
app.use((err, req, res, next) => {
  console.error(err.message); // Log the error message
  res.status(500).send(`An error occurred: ${err.message}`); // Send error response
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Route Specific Middleware

- Auth middleware will be called when a POST request is sent on '/users' route
- We can add properties into **req** object and can access in next middleware

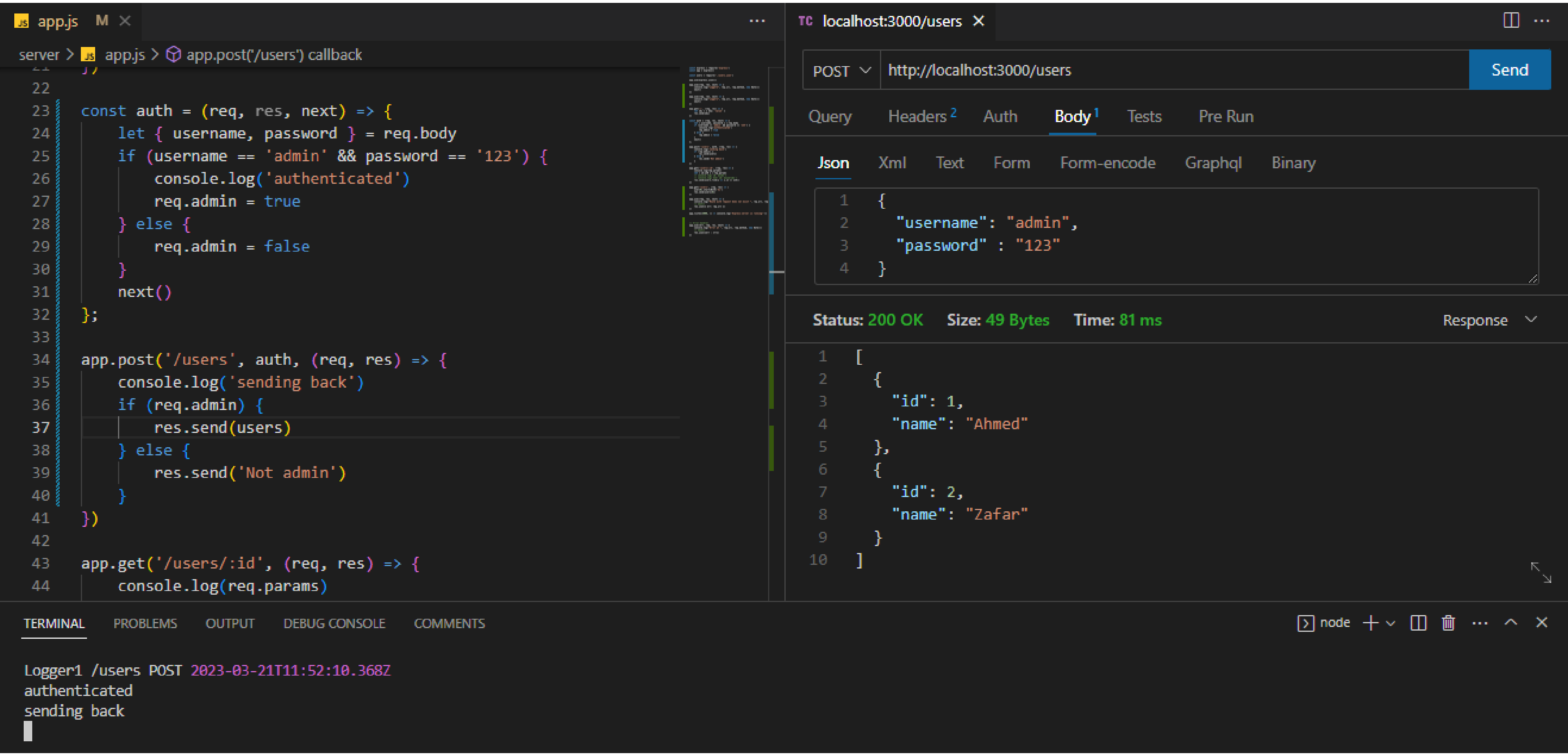


```
const auth = (req, res, next) => {
    let { username, password } = req.body
    if (username == 'admin' && password == '123') {
        console.log('authenticated')
        req.admin = true
    } else {
        req.admin = false
    }
    next()
};

app.post('/users', auth, (req, res) => {
    console.log('sending back')
    if (req.admin) {
        res.send(users)
    } else {
        res.send('Not admin')
    }
})
```

National University
Of Computer and Emerging Sciences

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

const authMiddleware = (req, res, next) => {
   const isAuthenticated = req.headers['authorization'] === 'Bearer secret-token';
   if (isAuthenticated) {
      next(); // User is authenticated, proceed to the next middleware or route} else {
      res.status(403).send('Forbidden: You are not authorized to access this resource.');
   }
};

// Public route that does not require authentication
app.get('/public', (req, res) => {
   res.send('This is a public route. Anyone can access this.');
});

// Protected route that requires authentication
app.get('/protected', authMiddleware, (req, res) => {
   res.send('This is a protected route. You are authorized to access this.');
});

app.listen(3000, () => {
   console.log('Server is running on port 3000');
});
```

Sample Request/Response (success)



```javascript
const auth = (req, res, next) => {
    let { username, password } = req.body
    if (username == 'admin' && password == '123') {
        console.log('authenticated')
        req.admin = true
    } else {
        req.admin = false
    }
    next()
};

app.post('/users', auth, (req, res) => {
    console.log('sending back')
    if (req.admin) {
        res.send(users)
    } else {
        res.send('Not admin')
    }
})

app.get('/users/:id', (req, res) => {
    console.log(req.params)
```

```json
{
  "username": "admin",
  "password" : "123"
}
```

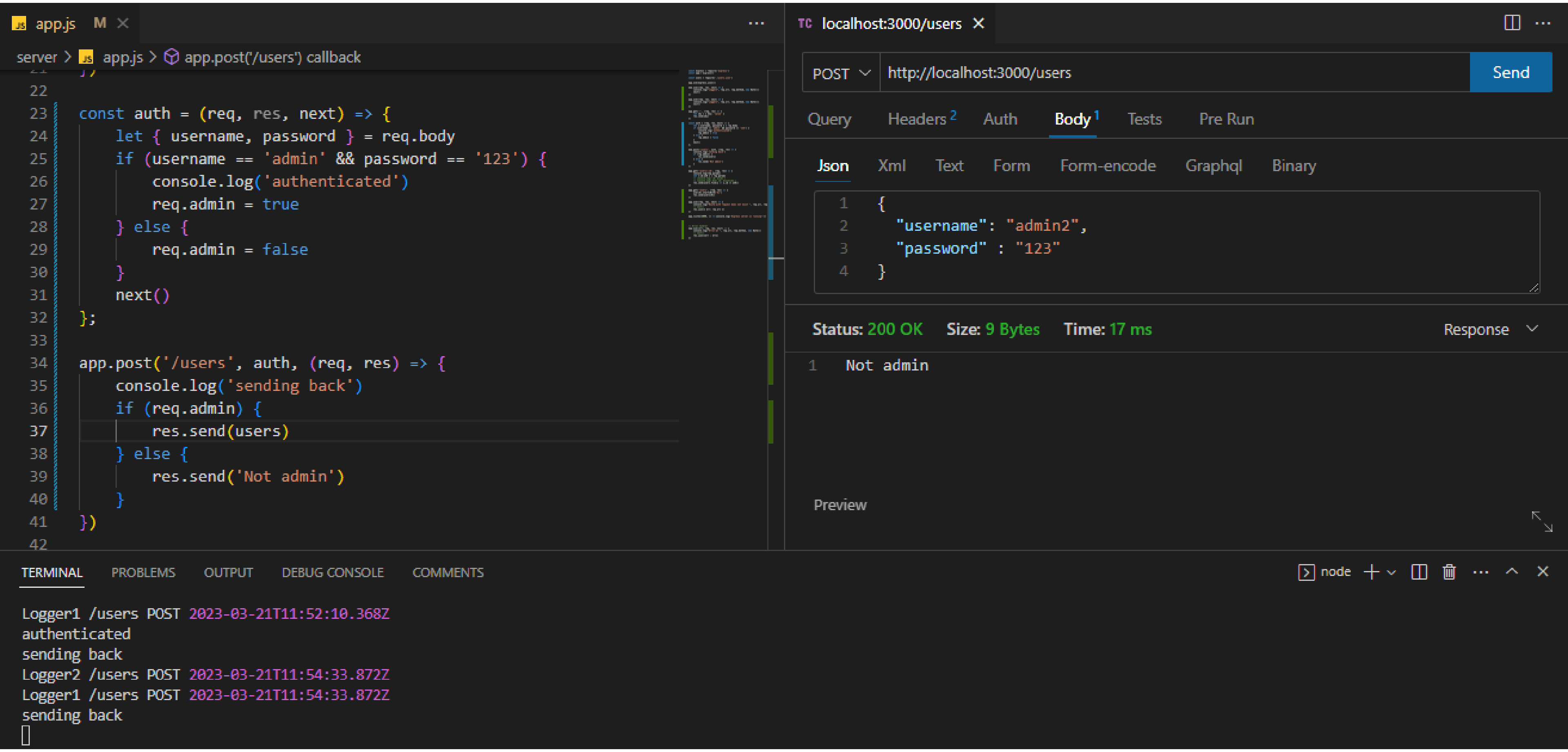
Status: 200 OK Size: 49 Bytes Time: 81 ms

```json
[
  {
    "id": 1,
    "name": "Ahmed"
  },
  {
    "id": 2,
    "name": "Zafar"
  }
]
```

```
Logger1 /users POST 2023-03-21T11:52:10.368Z
authenticated
sending back
```

National University
Of Computer and Emerging Sciences

Sample Request/Response (fail)



app.js M × | server > app.js > app.post('/users') callback

```js
const auth = (req, res, next) => {
    let { username, password } = req.body
    if (username == 'admin' && password == '123') {
        console.log('authenticated')
        req.admin = true
    } else {
        req.admin = false
    }
    next()
};

app.post('/users', auth, (req, res) => {
    console.log('sending back')
    if (req.admin) {
        res.send(users)
    } else {
        res.send('Not admin')
    }
})
```

POST http://localhost:3000/users Send

Body (Json):

```json
{
  "username": "admin2",
  "password" : "123"
}
```

Status: 200 OK Size: 9 Bytes Time: 17 ms

```
Not admin
```

TERMINAL

```
Logger1 /users POST 2023-03-21T11:52:10.368Z
authenticated
sending back
Logger2 /users POST 2023-03-21T11:54:33.872Z
Logger1 /users POST 2023-03-21T11:54:33.872Z
sending back
```

National University
Of Computer and Emerging Sciences

Router Middleware

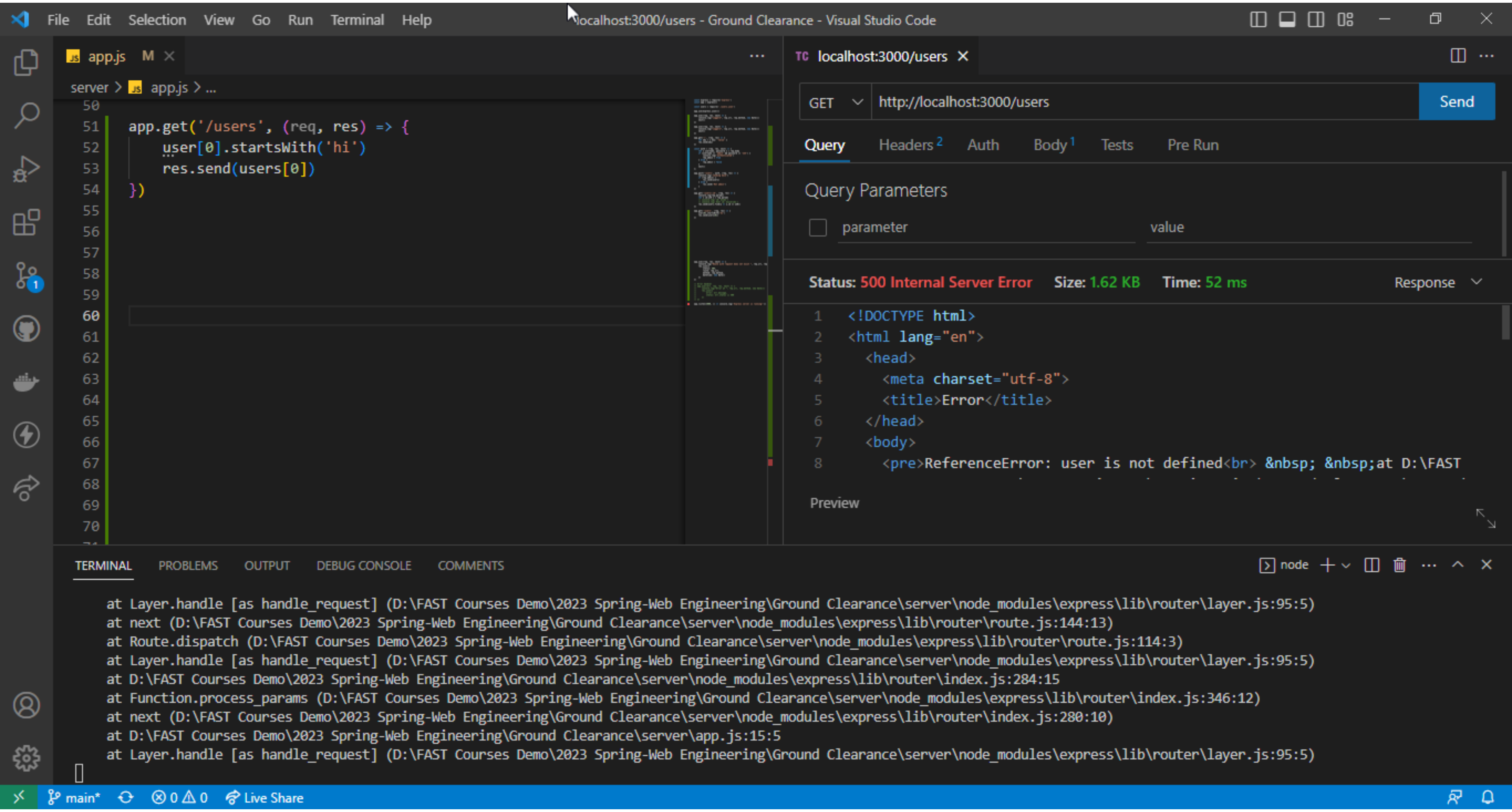
- Create a simple request logger middleware that prints out request URL, method, and time
  - The `next` argument
  - Add the middleware to the application using `app.use()`
  - Middleware can also be added at the router level with `router.use()`

```
const express = require('express');
const app = express();
const router = express.Router();

const authMiddleware = (req, res, next) => {
  const isAuthenticated = req.headers['authorization'] === 'Bearer secret-token';
  if (isAuthenticated) {
    next(); // User is authenticated, proceed to the next middleware or route} else {
    res.status(403).send('Forbidden: You are not authorized to access this resource.');
  }};

router.get('/public', (req, res) => {
  res.send('This is a public route. Anyone can access this.');
});
router.get('/protected', authMiddleware, (req, res) => {
  res.send('This is a protected route. You are authorized to access this.');
});

app.use('/', router);

app.listen(3000, () => {
  console.log('Server is running on port 3000');});
```

```
// routes.js
const express = require('express');
const router = express.Router();

const authMiddleware = (req, res, next) => {
  const isAuthenticated = req.headers['authorization'] === 'Bearer secret-token';

  if (isAuthenticated) {
    next(); // User is authenticated, proceed to the next middleware or route
  } else {
    res.status(403).send('Forbidden: You are not authorized to access this resource.');
  }
};

router.get('/public', (req, res) => {
  res.send('This is a public route. Anyone can access this.');
});
router.get('/protected', authMiddleware, (req, res) => {
  res.send('This is a protected route. You are authorized to access this.');
});

module.exports = router;
```

```
// app.js
const express = require('express');
const app = express();
const routes = require('./routes'); // Importing the router from routes.js

// Use the router in the app
app.use('/', routes); // Mount the router to the root path

// Start the server
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

App Crashes on Error


File Edit Selection View Go Run Terminal Help
localhost:3000/users - Ground Clearance - Visual Studio Code
app.js M
server > app.js > ...
localhost:3000/users
GET
Send
Query Headers 2 Auth Body 1 Tests Pre Run
Query Parameters
parameter
value
Response
Preview
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE COMMENTS
node
main* ⊗ 0 ⚠ 0 Live Share

```
app.get('/users', (req, res) => {
    user[0].startsWith('hi')
    res.send(users[0])
})
```

http://localhost:3000/users

Status: 500 Internal Server Error Size: 1.62 KB Time: 52 ms

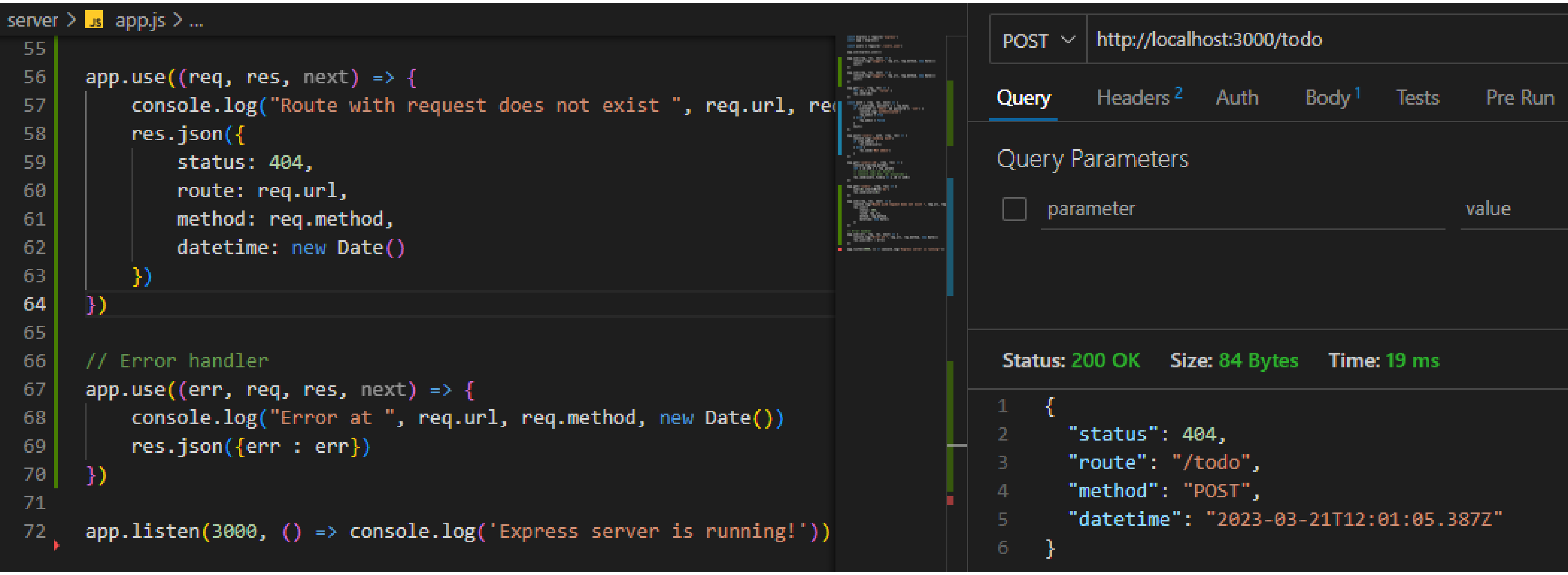
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Error</title>
  </head>
  <body>
    <pre>ReferenceError: user is not defined<br> &nbsp; &nbsp;at D:\FAST
```

```
    at Layer.handle [as handle_request] (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\layer.js:95:5)
    at next (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\route.js:144:13)
    at Route.dispatch (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\route.js:114:3)
    at Layer.handle [as handle_request] (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\layer.js:95:5)
    at D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\index.js:284:15
    at Function.process_params (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\index.js:346:12)
    at next (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\index.js:280:10)
    at D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\app.js:15:5
    at Layer.handle [as handle_request] (D:\FAST Courses Demo\2023 Spring-Web Engineering\Ground Clearance\server\node_modules\express\lib\router\layer.js:95:5)
```

National University
Of Computer and Emerging Sciences

Avoid Default Error Page and Error Handling Middleware

- To avoid default error page
- To avoid app crashing
- Add these middleware at the end of all requests



server > app.js > ...

```javascript

app.use((req, res, next) => {
    console.log("Route with request does not exist ", req.url, req
    res.json({
        status: 404,
        route: req.url,
        method: req.method,
        datetime: new Date()
    })
})

// Error handler
app.use((err, req, res, next) => {
    console.log("Error at ", req.url, req.method, new Date())
    res.json({err : err})
})

app.listen(3000, () => console.log('Express server is running!'))
```

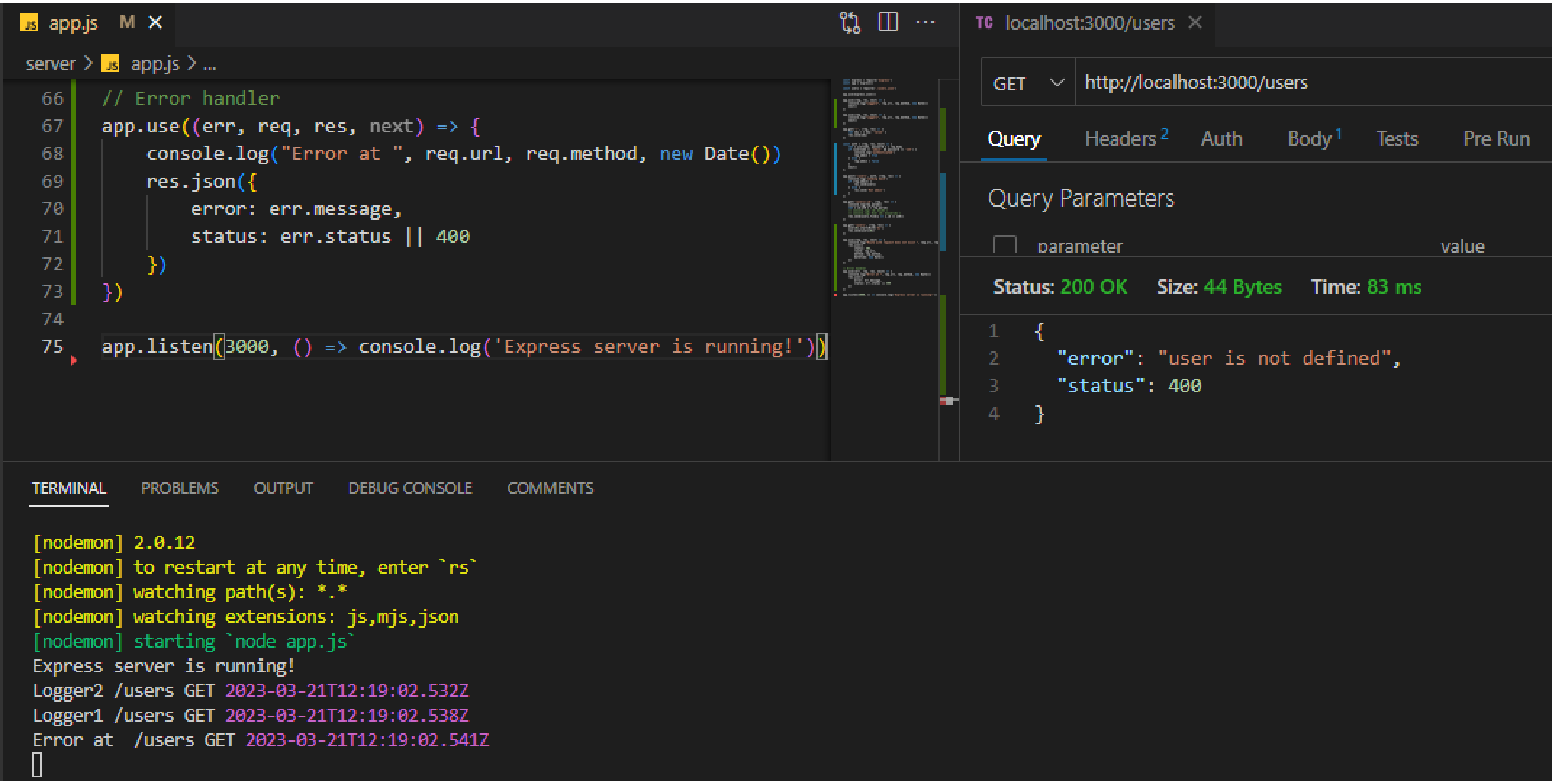
POST http://localhost:3000/todo

Query Parameters: parameter | value

Status: 200 OK   Size: 84 Bytes   Time: 19 ms

```json
{
  "status": 404,
  "route": "/todo",
  "method": "POST",
  "datetime": "2023-03-21T12:01:05.387Z"
}
```

Error Handling Middleware



app.js M ×

server > app.js > ...

```javascript
// Error handler
app.use((err, req, res, next) => {
    console.log("Error at ", req.url, req.method, new Date())
    res.json({
        error: err.message,
        status: err.status || 400
    })
})

app.listen(3000, () => console.log('Express server is running!'))
```

localhost:3000/users ×

GET http://localhost:3000/users

Query | Headers 2 | Auth | Body 1 | Tests | Pre Run

Query Parameters

parameter value

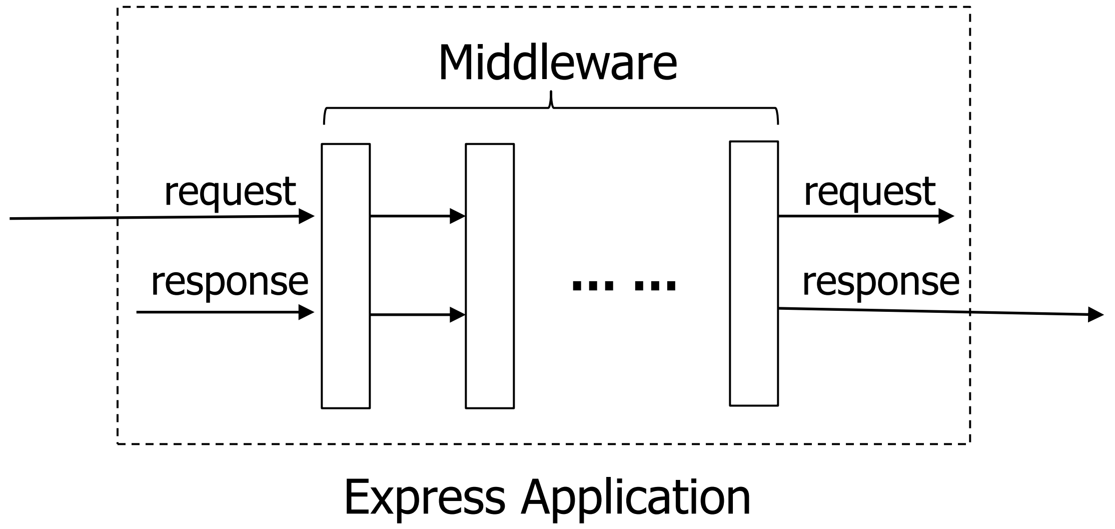
Status: 200 OK Size: 44 Bytes Time: 83 ms

```json
{
  "error": "user is not defined",
  "status": 400
}
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE COMMENTS

```
[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Express server is running!
Logger2 /users GET 2023-03-21T12:19:02.532Z
Logger1 /users GET 2023-03-21T12:19:02.538Z
Error at  /users GET 2023-03-21T12:19:02.541Z
```

Middleware



- A middleware is a function that has access to three arguments: the `request` object, the `response` object, and a `next` function that passes control to the next middleware function

Other Middlewares

- `express.json()` parses JSON request body and add JSON object properties to `req.body`
- `express.urlencoded()` parses urlencoded request body and request parameters to `req.body`
- Route handler functions are also middleware
  - Where is `next`??
  - Remember to use `next` if you have more than one handler functions for a route
- *Middleware order is important!*

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

// Use express.json() to parse JSON bodies
app.use(express.json());

app.post('/json-data', (req, res) => {
    console.log(req.body); // Access parsed JSON data
    res.send(`Received JSON data: ${JSON.stringify(req.body)}`);
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

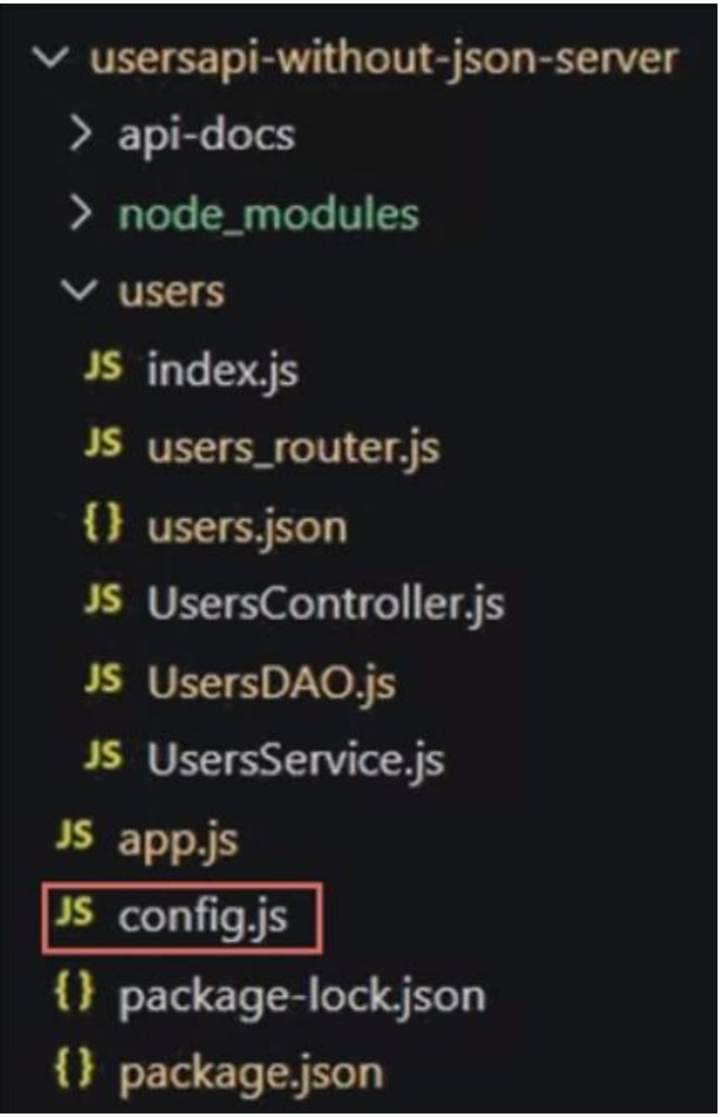
```
const express = require('express');
const app = express();

// Use express.urlencoded() to parse URL-encoded bodies
app.use(express.urlencoded({ extended: true }));

app.post('/form-data', (req, res) => {
        const { name, age } = req.body;
    res.send(`Received data: Name is ${name} and Age is ${age}`);
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

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

app.post('/submit-form', express.urlencoded({ extended: true }), (req, res) =>
{
      const { name, age } = req.body;
      res.send(`Received data: Name is ${name} and Age is ${age}`);
});

app.listen(3000, () => {
   console.log('Server is running on port 3000');
});
```

Nodejs Application Structure

Routes	• Forward the request to appropriate controller functions • To make the code modular, use the command: • `const express = require('express');` • `const router = express.Router();` • Route handlers can be defined separately in a `.js` file instead of an `app.js` file.
Controller	Callback functions passed to the router methods
Service Layer	Handles the business logic of the application
DAO Layer	Used to perform operations on the data resource

Nodejs Application Structure



```
∨ usersapi-without-json-server
  > api-docs
  > node_modules
  ∨ users
    JS index.js
    JS users_router.js
    {} users.json
    JS UsersController.js
    JS UsersDAO.js
    JS UsersService.js
  JS app.js
  JS config.js
  {} package-lock.json
  {} package.json
```

`app.js` is the entry point for the application and calls `index.js` for the routes.

The `index.js` file references the `users_router.js`.

The `users_router.js` file contains all the routes.

`users.json` consists of data about the users.

`UsersDAO.js` performs all manipulation operations on the data.

`UsersService.js` contains code to perform all the business logic.

`UsersController.js` handles incoming requests and returns responses.

`config.js` consists of configuration details.

Readings

- Express Documentation