Middle wares –

Inline middlewares –

You can use inline middlewares directly by passing it to the get/post/delete/etc. in the order that you want it to be executed in like in line no. 7 below -



**What Are Middlewares in Express?**

In **Express.js**, middleware refers to functions that have access to the **request object (req)**, the **response object (res)**, and the **next middleware function in the application’s request-response cycle**. Middleware functions can perform the following tasks:

* **Execute any code.**
* **Modify the request and response objects.**
* **End the request-response cycle.**
* **Call the next middleware in the stack.**

Middleware functions are a central part of Express applications, as they help in processing and handling incoming requests.

**How Are Middlewares Used in Express?**

Middlewares are used by **defining them in the request-response cycle** and specifying when they should run. They can be applied globally to all routes or to specific routes.

1. **Defining a Middleware:**

javascript

CopyEdit

const express = require('express');

const app = express();

// Example middleware

const logger = (req, res, next) => {

console.log(`${req.method} ${req.url}`);

next(); // Pass control to the next middleware

};

app.use(logger); // Attach the middleware globally

1. **Types of Middleware:**
   * **Application-Level Middleware:** Applied to the entire app or specific routes.

javascript

CopyEdit

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

console.log('Global middleware');

next();

});

app.get('/user', (req, res) => {

res.send('User route');

});

* + **Router-Level Middleware:** Specific to routers.

javascript

CopyEdit

const router = express.Router();

router.use((req, res, next) => {

console.log('Router-specific middleware');

next();

});

router.get('/profile', (req, res) => {

res.send('Profile page');

});

app.use('/user', router);

* + **Built-In Middleware:** Provided by Express (e.g., express.json()).

javascript

CopyEdit

app.use(express.json());

* + **Third-Party Middleware:** Installed via npm (e.g., body-parser, cors).

javascript

CopyEdit

const cors = require('cors');

app.use(cors());

* + **Error-Handling Middleware:** Specialized middleware for handling errors.

javascript

CopyEdit

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

console.error(err.stack);

res.status(500).send('Something went wrong!');

});

**What Are Inline Middlewares?**

Inline middleware refers to middleware functions that are **directly defined within a route or an API call**, instead of being declared separately. These are commonly used for simpler, one-time logic.

Example:

javascript

CopyEdit

app.get('/user', (req, res, next) => {

console.log('Inline middleware for /user');

next(); // Pass control to the next middleware or route handler

}, (req, res) => {

res.send('User endpoint response');

});

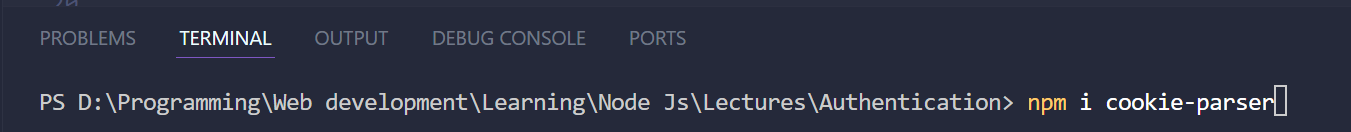
In this example:

* The first function logs a message and calls next().
* The second function sends the response.

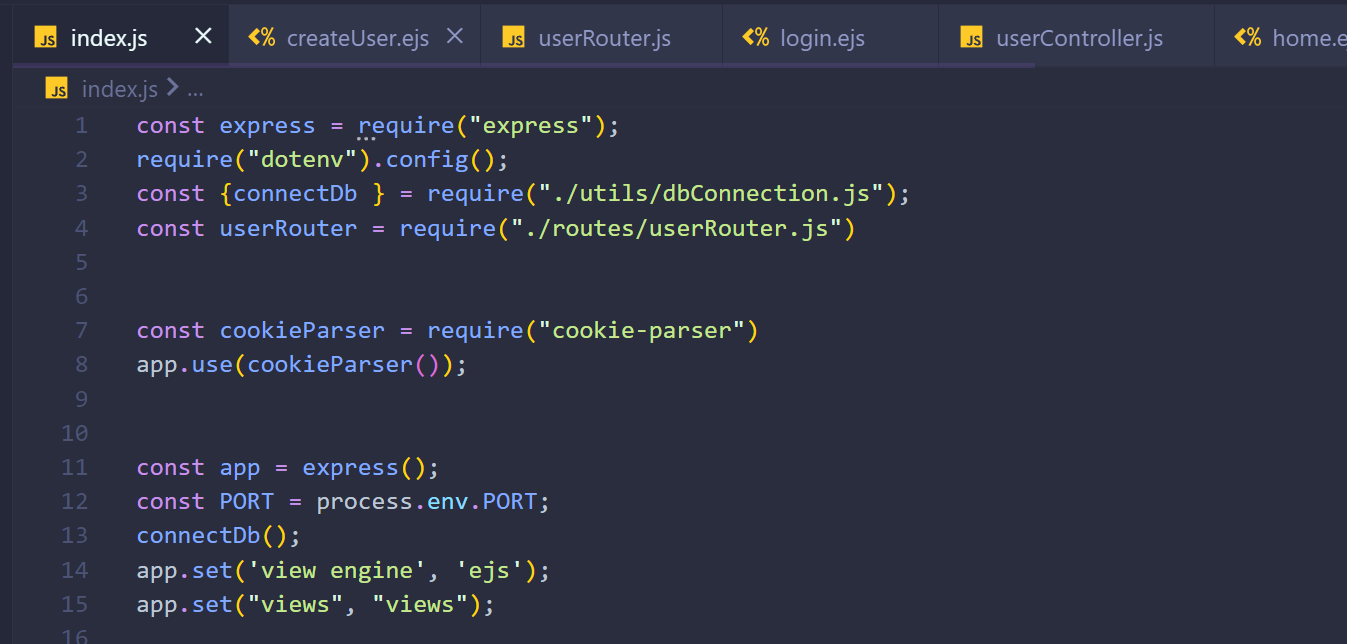
**Summary:**

* Middlewares are functions in Express that process requests and responses.
* They can be used globally or specifically for routes.
* Inline middleware refers to middleware defined directly in route definitions.
* They are key to handling tasks like logging, authentication, and error handling in Express applications.

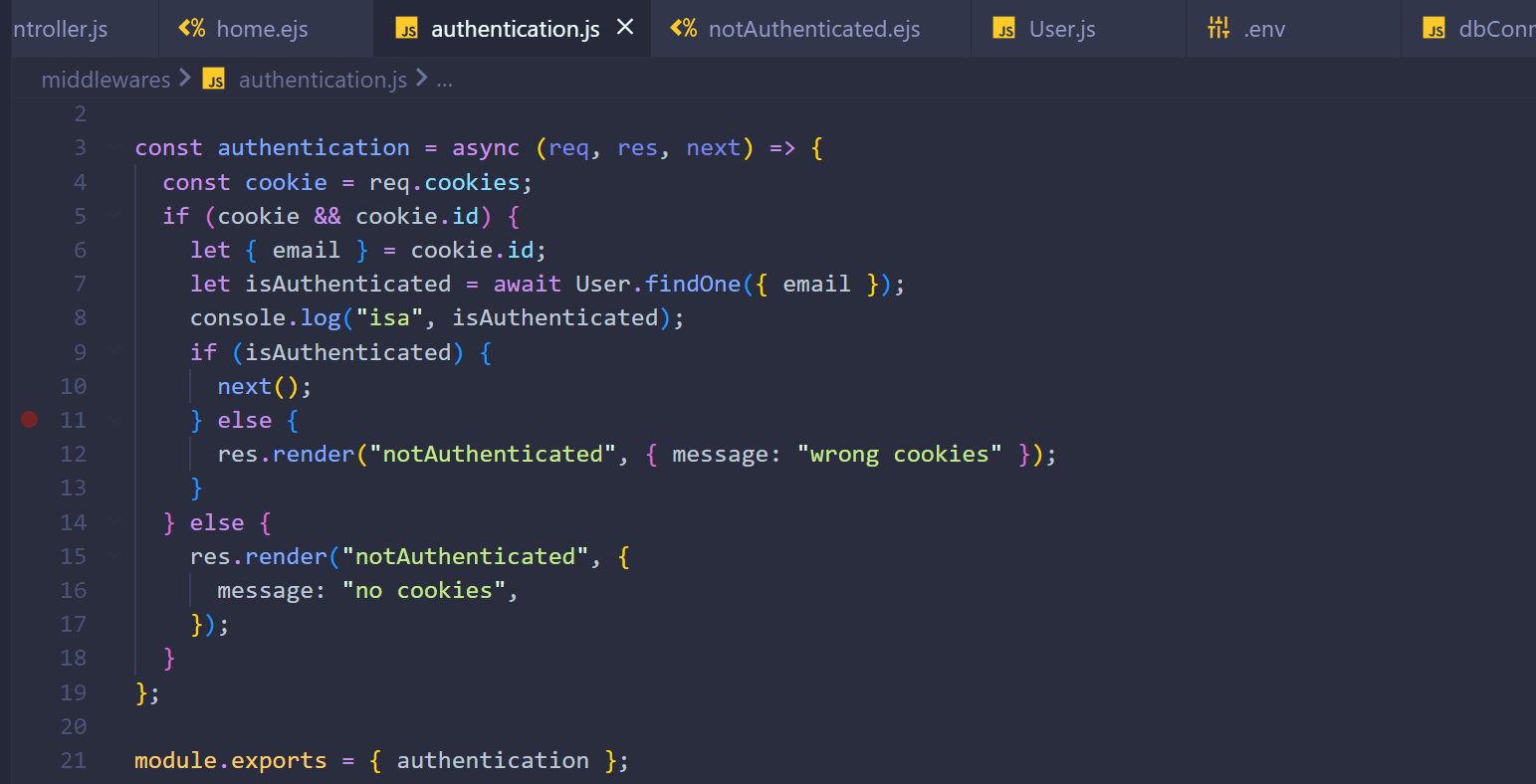
Cookies –



Import and use as a middleware –



Using cookies –



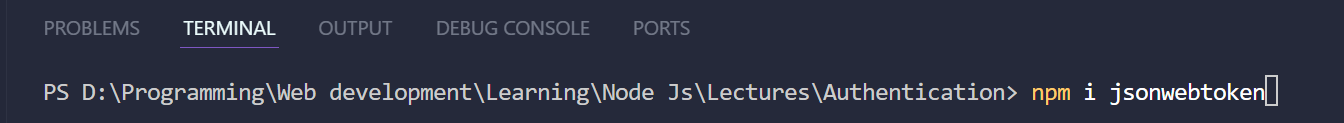
Setting and removing cookies -

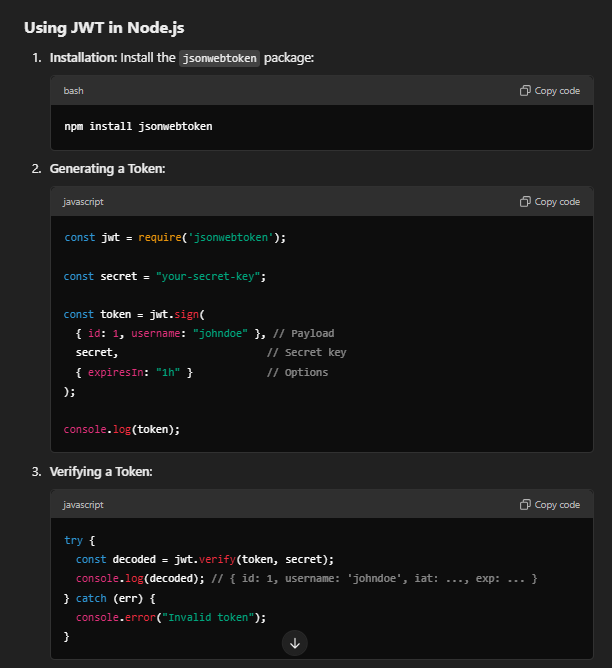


JWT Tokens –

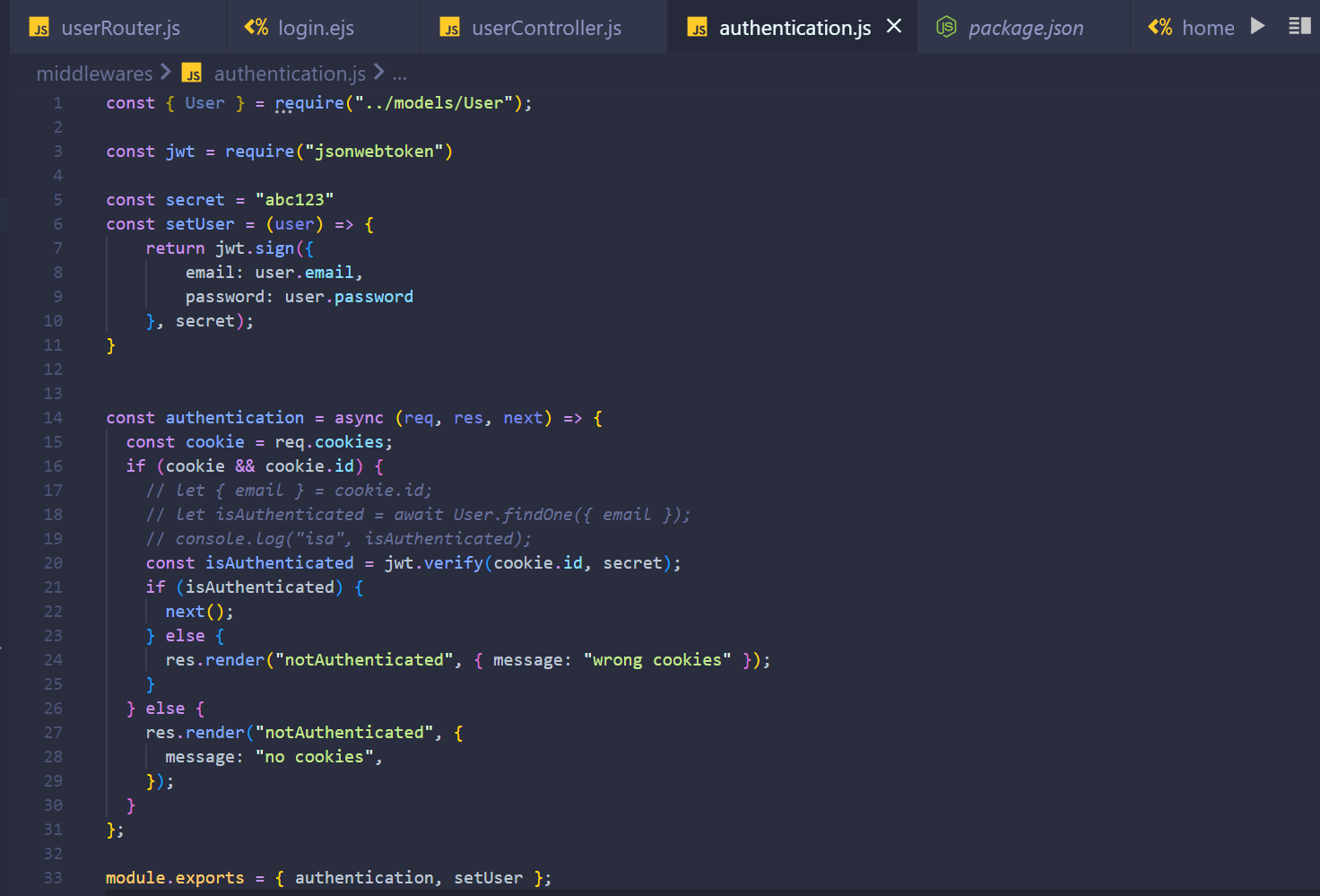
Used for stateless authentication

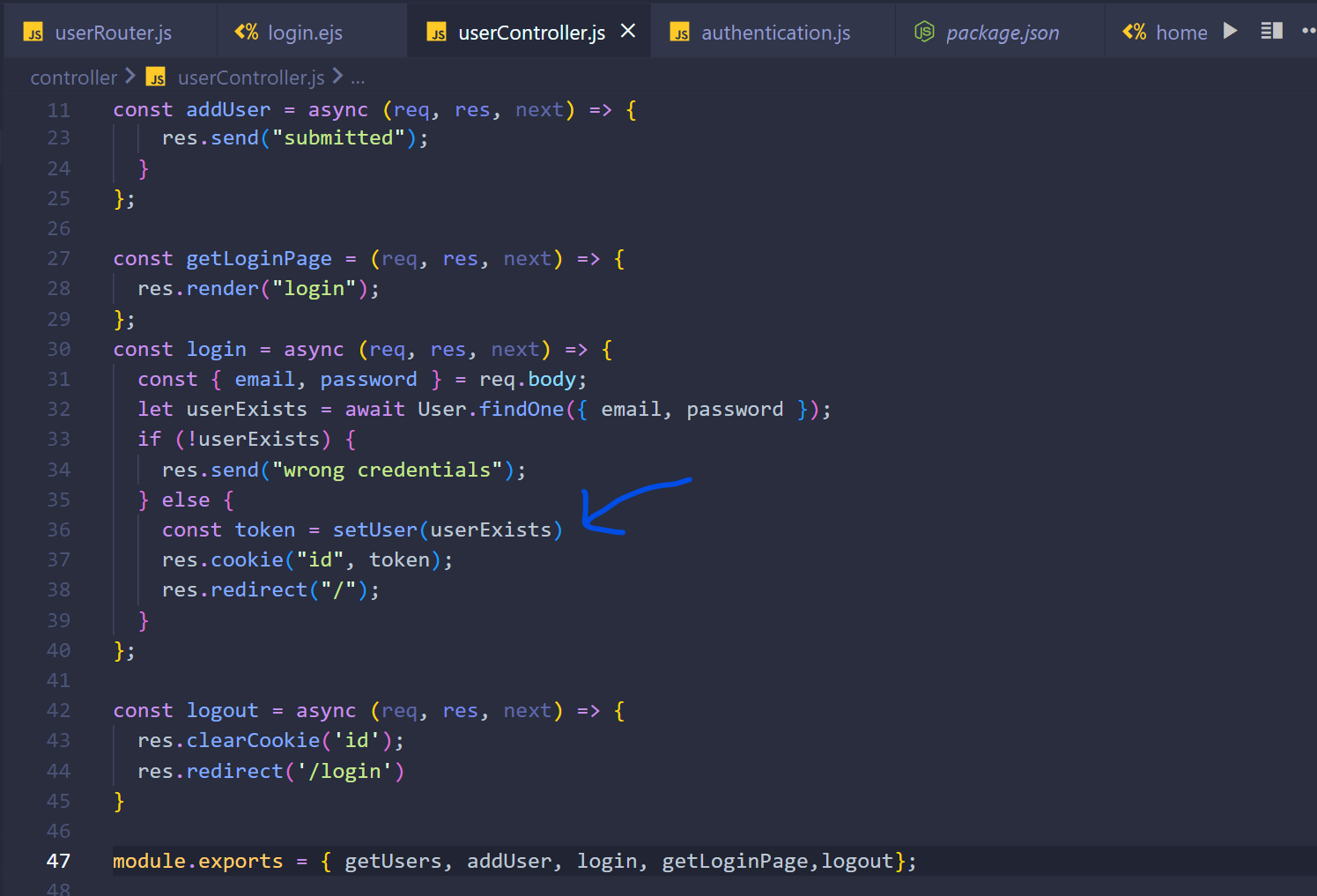
Install -





Using jwt tokens –





Optional chaining operator –

The syntax you're referring to is the **optional chaining operator (?.)** introduced in JavaScript (ES2020). It allows you to safely access deeply nested properties or methods of an object without having to check each level for null or undefined.

**How It Works:**

The ?. operator short-circuits and returns undefined if the value before it is null or undefined. This prevents errors like Cannot read property 'x' of undefined.

**Examples:**

**Accessing Nested Properties**

Without ?.:

javascript

Copy code

const user = { profile: { name: "John" } };

console.log(user.profile.name); // "John"

// console.log(user.profile.age); // Undefined

// console.log(user.address.city); // Error: Cannot read property 'city' of undefined

With ?.:

javascript

Copy code

const user = { profile: { name: "John" } };

console.log(user?.profile?.name); // "John"

console.log(user?.profile?.age); // undefined (no error)

console.log(user?.address?.city); // undefined (no error)

**Accessing Methods**

Without ?.:

javascript

Copy code

const user = { getName: () => "John" };

console.log(user.getName()); // "John"

// console.log(user.getAge()); // Error: user.getAge is not a function

With ?.:

javascript

Copy code

const user = { getName: () => "John" };

console.log(user.getName?.()); // "John"

console.log(user.getAge?.()); // undefined (no error)

**Arrays and Optional Chaining**

javascript

Copy code

const users = [{ name: "John" }, null];

console.log(users[0]?.name); // "John"

console.log(users[1]?.name); // undefined (no error)

**Use Cases:**

1. **Safely Access Nested Object Properties**: Useful when working with APIs or data where certain fields may not always exist.

javascript

Copy code

const response = {

data: {

user: null,

},

};

console.log(response?.data?.user?.name); // undefined (instead of throwing an error)

1. **Safely Invoke Methods**: Ensure the method exists before calling it.

javascript

Copy code

const obj = {

greet: () => "Hello!",

};

console.log(obj.greet?.()); // "Hello!"

console.log(obj.sayGoodbye?.()); // undefined (no error)

1. **Check Array Items**: Safely access properties of items in an array.

javascript

Copy code

const items = [null, { id: 1 }];

console.log(items[0]?.id); // undefined (no error)

console.log(items[1]?.id); // 1

1. **Dynamic Properties**: Use variables with optional chaining.

javascript

Copy code

const key = "profile";

const user = { profile: { name: "John" } };

console.log(user?.[key]?.name); // "John"

console.log(user?.[key]?.age); // undefined

**Why Use It?**

* **Cleaner code**: Avoid long chains of if or && conditions.
* **Prevent runtime errors**: Helps avoid Cannot read property 'x' of undefined errors.
* **Safe and readable**: Particularly useful when working with APIs or unstructured data.