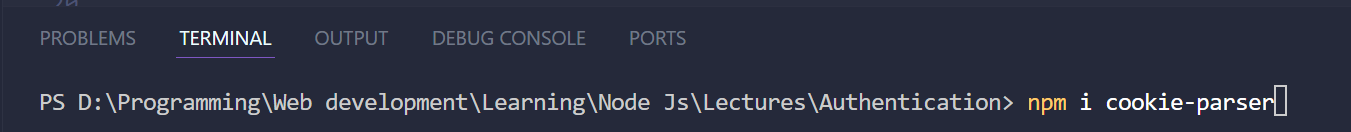
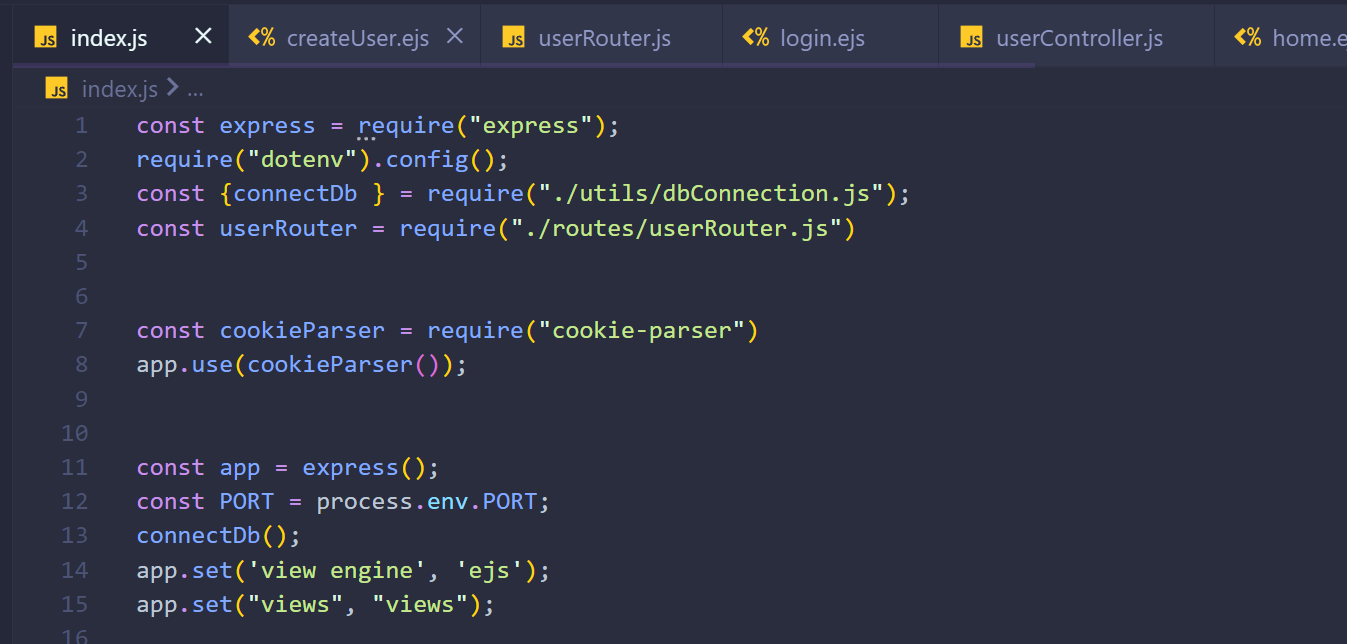
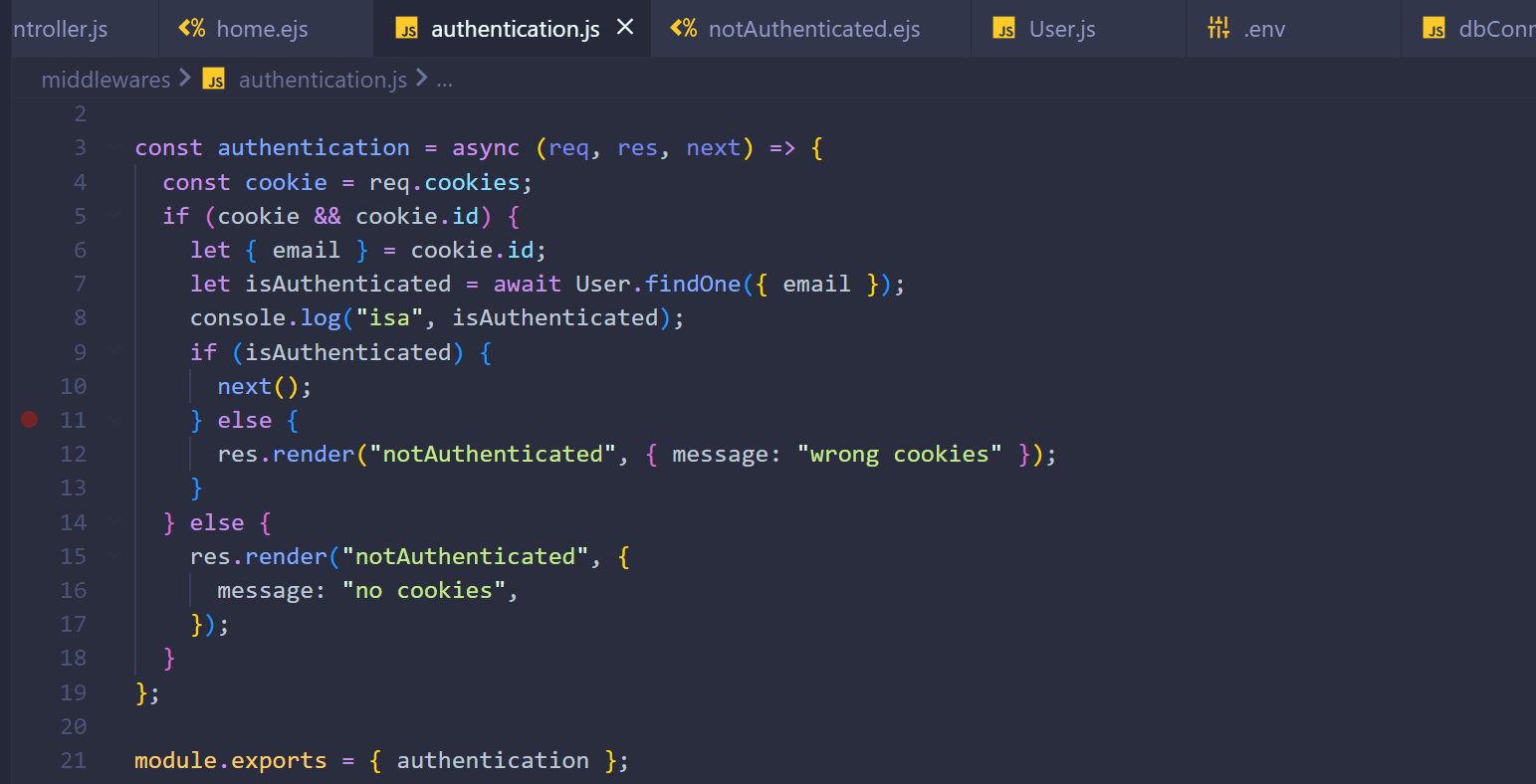
Cookies –



Import and use as a middleware –



Using cookies –



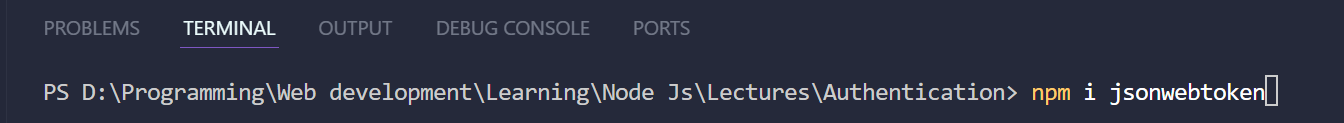
Setting and removing cookies -

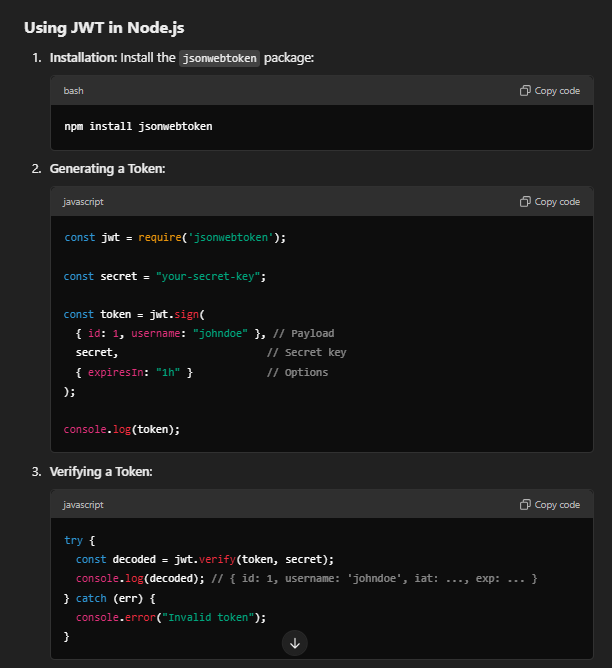


JWT Tokens –

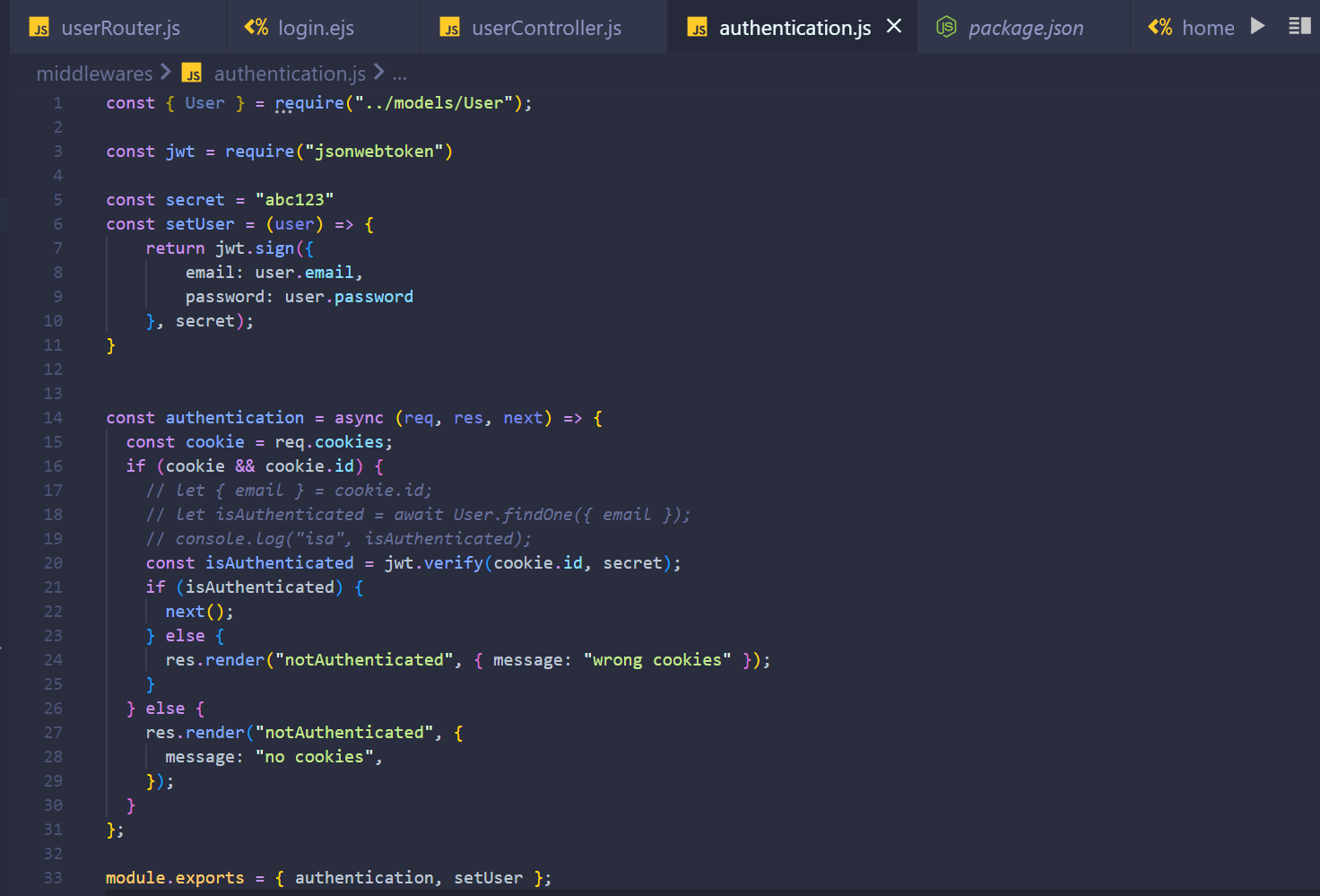
Used for stateless authentication

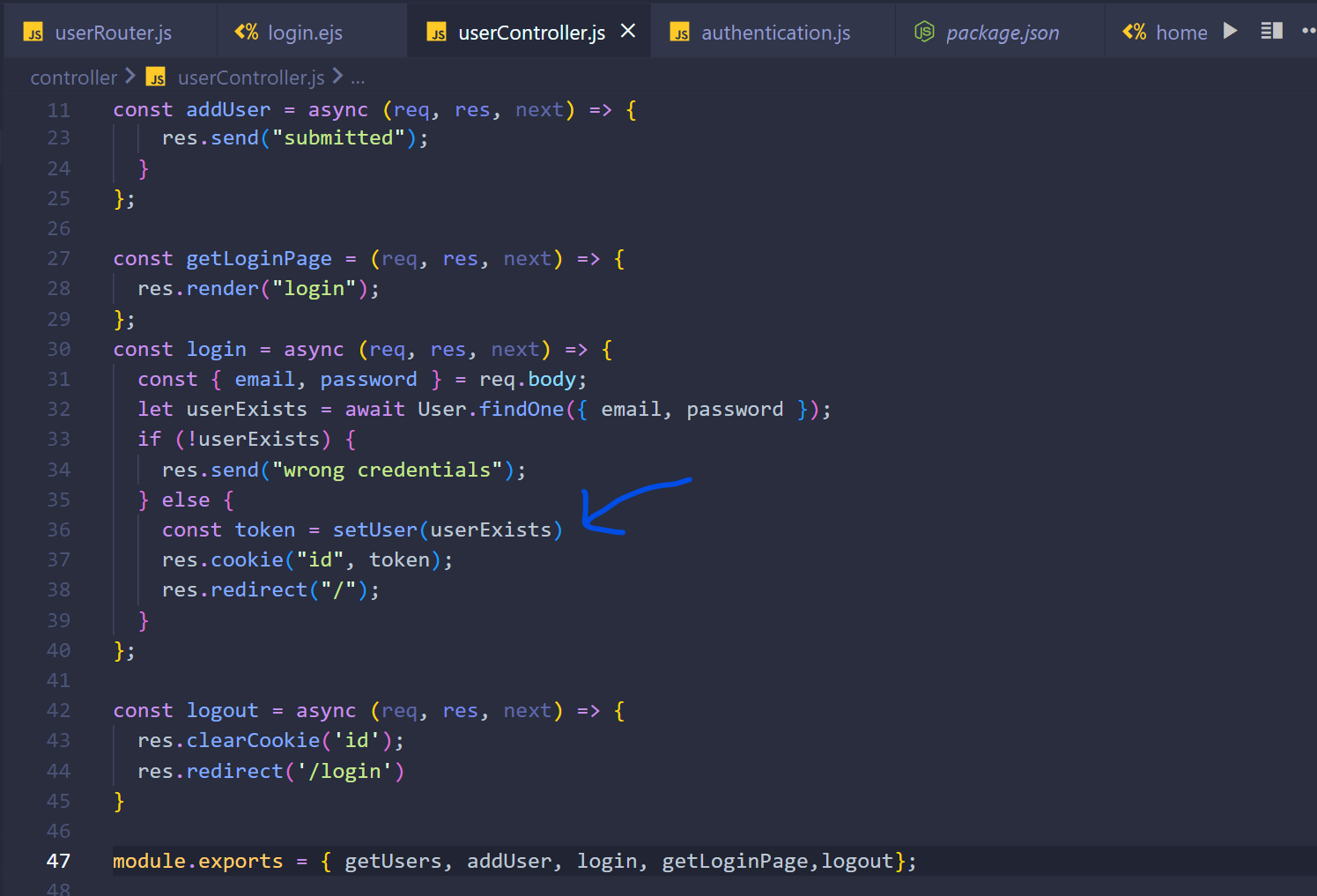
Install -





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.