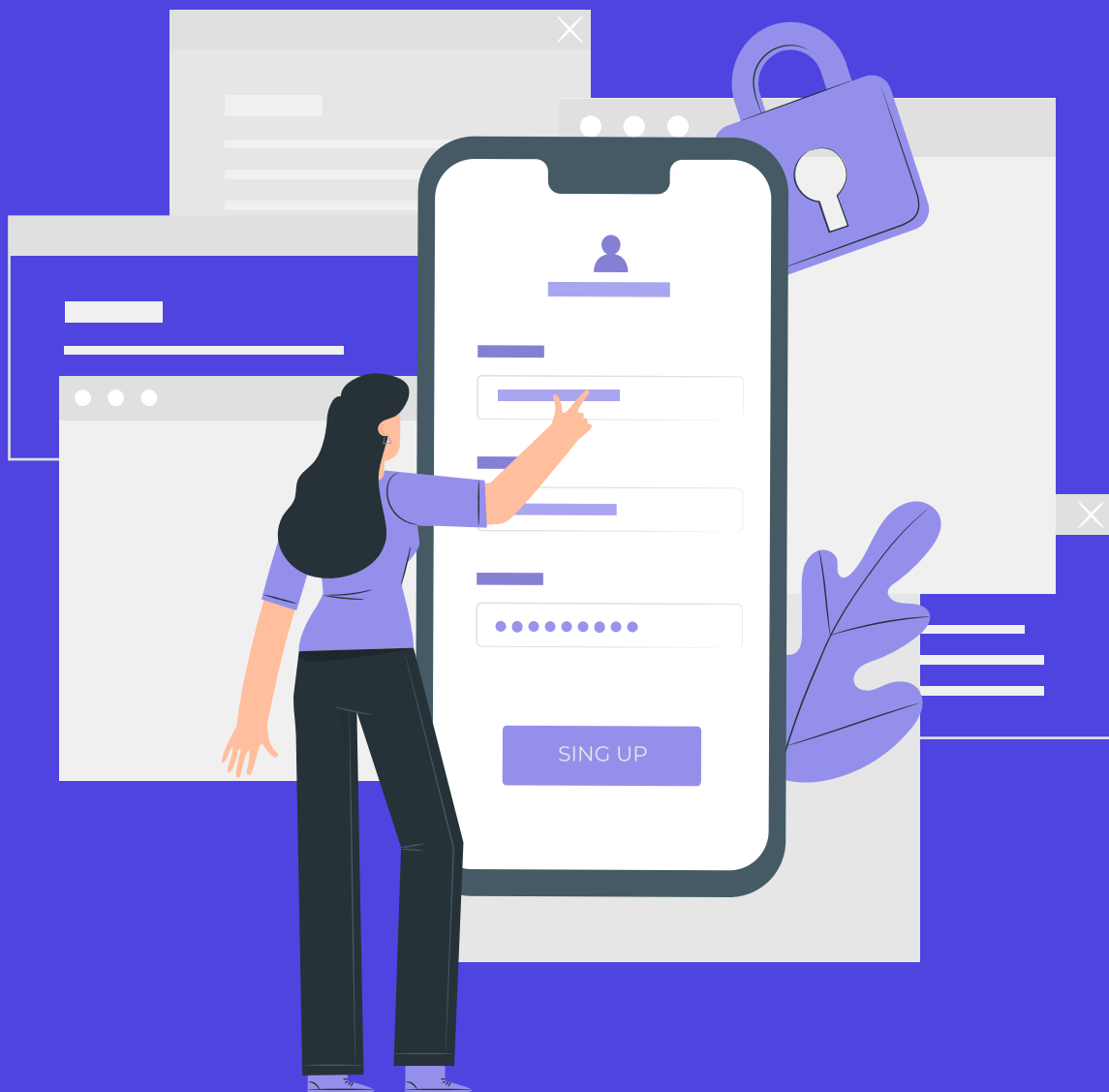


Filters and validations of request using middlewares Assignment Solution



Let us continue with our mini project and add some authentication which we learnt in today's class.

1. Create User Table, Schema and corresponding route and a controller with login and register functions.

Step 1: Let's create users table in our books_db.

The screenshot shows the MySQL Workbench interface. The left sidebar contains a 'MANAGEMENT' section with options like Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, and Data Import/Restore. Below this is an 'INSTANCE' section with Startup / Shutdown, Server Logs, and Options File. The 'PERFORMANCE' section includes Dashboard, Performance Reports, and Performance Schema Setup. The main editor window is titled 'books_db*' and contains the following SQL code:

```
1 • use books_db;
2
3 • create table users(
4     id INT NOT NULL AUTO_INCREMENT,
5     username VARCHAR(30) NOT NULL,
6     email VARCHAR(30) NOT NULL ,
7     password VARCHAR(100) NOT NULL,
8     createdAt DATE,
9     updatedAt DATE,
10    PRIMARY KEY ( id )
11 );
12
13 • select * from users;
14
```

Below the SQL editor, the 'Result Grid' is displayed, showing the output of the query. It has columns for id, username, email, password, createdAt, and updatedAt. The first row shows a user with id 1, username 'mohit', email 'mohit@test.com', and a hashed password. The second row shows a user with id NULL, username NULL, email NULL, password NULL, createdAt NULL, and updatedAt NULL.

id	username	email	password	createdAt	updatedAt
1	mohit	mohit@test.com	\$2a\$08\$rnPo2XRC5OI4/xhIFhVO9KIhwB8FLJ...	2022-02-22	2022-02-22
NULL	NULL	NULL	NULL	NULL	NULL

The bottom of the interface shows a 'users 4 x' tab with 'Apply' and 'Revert' buttons. The 'Output' section at the bottom is currently empty.

The screenshot displays the Visual Studio Code interface with three main panels:

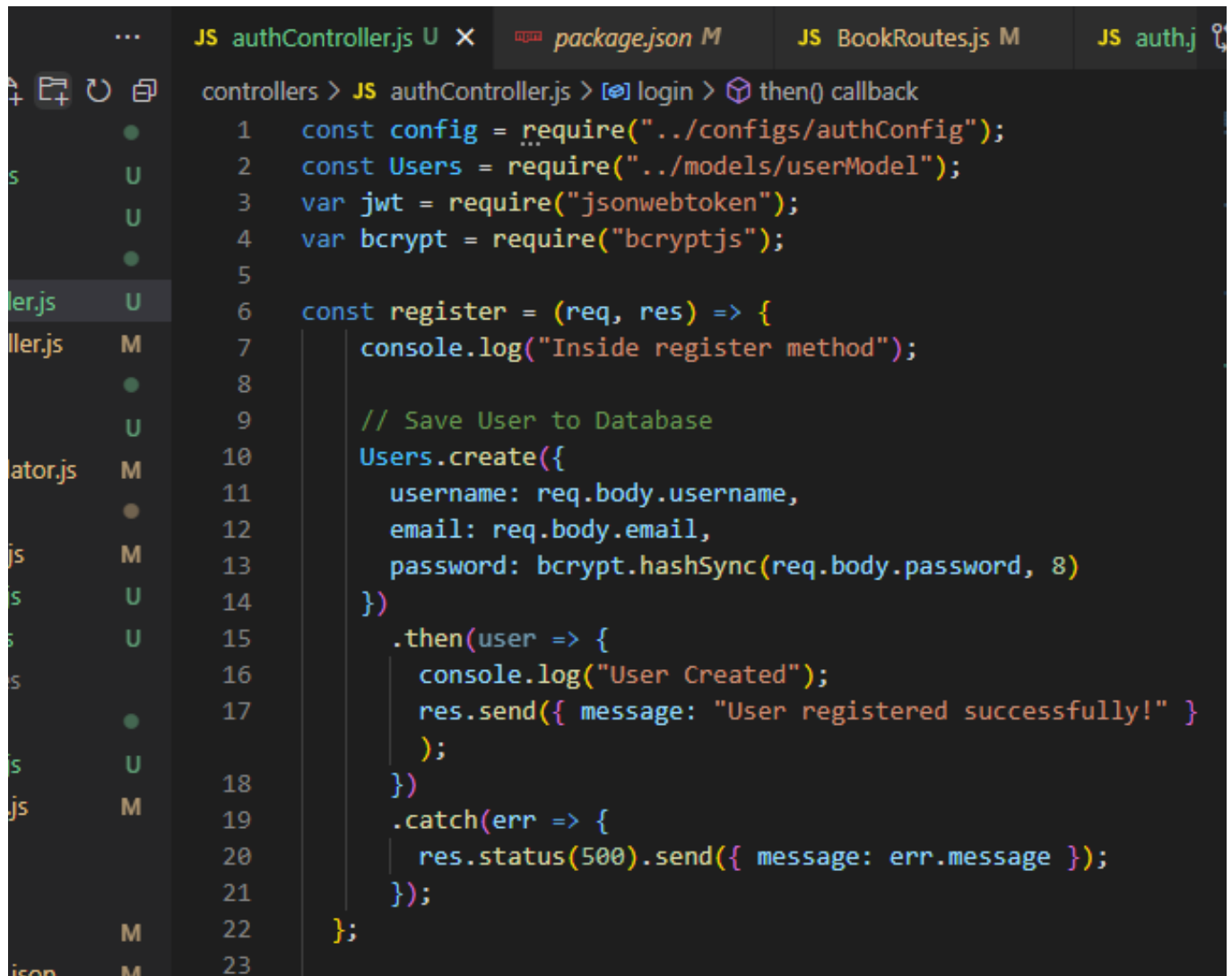
- EXPLORER:** Shows the project structure for 'BASIC-EXPRESS-APP'. The 'models' directory is expanded, showing 'bookModel.js', 'booksStore.js', and 'userModel.js' (selected).
- EDITOR:** Displays the code for 'models > JS userModel.js'. The code defines a Sequelize model for 'Users' with fields 'username', 'email', and 'password', all of type 'DataTypes.STRING'. It also sets 'freezeTableName: true' and exports the model as 'Users'.
- TERMINAL:** Shows the command 'node' and the output 'Connection has been established successfully. Executing (default): UPDATE `books` SET `title`=?,`updatedAt`=? WHERE `id` = ?'.

Step 3: Before start creating authController.js, install jsonwebtoken and bcryptjs packages.

```
package.json > {} dependencies > mysql2

1  {
2    "name": "my-first-backend-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start:dev": "nodemon index.js",
8      "start": "node index.js",
9      "test": "echo \"Error: no test specified\" && exit
10   },
11   "keywords": [],
12   "author": "",
13   "license": "ISC",
14   "dependencies": {
15     "bcryptjs": "^2.4.3",
16     "body-parser": "^1.19.1",
17     "express": "^4.17.2",
18     "jsonwebtoken": "^8.5.1",
19     "mysql2": "^2.3.3",
20     "sequelize": "^6.16.2",
21     "uuid": "^8.3.2"
22   },
23   "devDependencies": {
24     "nodemon": "^2.0.15"
25   }
26 }
27
```

Step 4: Create authController.js in controllers folder with login and register functions

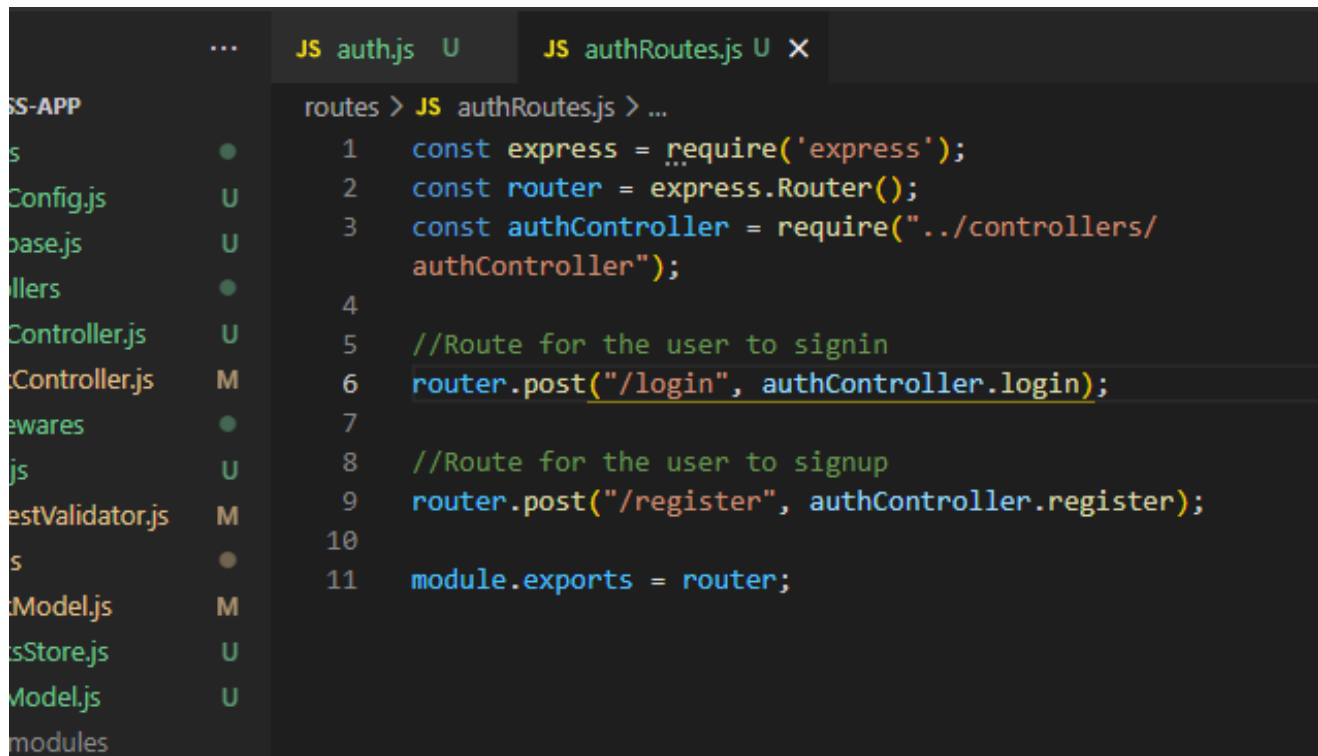


```
controllers > JS authController.js > login > then() callback
1  const config = require("../configs/authConfig");
2  const Users = require("../models/userModel");
3  var jwt = require("jsonwebtoken");
4  var bcrypt = require("bcryptjs");
5
6  const register = (req, res) => {
7      console.log("Inside register method");
8
9      // Save User to Database
10     Users.create({
11         username: req.body.username,
12         email: req.body.email,
13         password: bcrypt.hashSync(req.body.password, 8)
14     })
15     .then(user => {
16         console.log("User Created");
17         res.send({ message: "User registered successfully!" });
18     })
19     .catch(err => {
20         res.status(500).send({ message: err.message });
21     });
22 };
23
```

controllers > JS authController.js > login > then() callback

```
24 const login = (req, res) => {
25   Users.findOne({
26     where: {
27       username: req.body.username
28     }
29   })
30   .then(user => {
31     if (!user) {
32       return res.status(404).send({ message: "User Not
33         found." });
34     }
35     var passwordIsValid = bcrypt.compareSync(
36       req.body.password,
37       user.password
38     );
39     if (!passwordIsValid) {
40       return res.status(401).send({
41         accessToken: null,
42         message: "Invalid Password!"
43       });
44     }
45   })
46   .then(() => {
47     var token = jwt.sign({ id: user.id }, config.secret,
48       {
49         expiresIn: 86400 // 24 hours
50       });
51     res.status(200).send({
52       id: user.id,
53       username: user.username,
54       email: user.email,
55       accessToken: token
56     });
57   })
58   .catch(err => {
59     res.status(500).send({ message: err.message });
60   });
61 };
62
63
64 module.exports = { login, register }
```

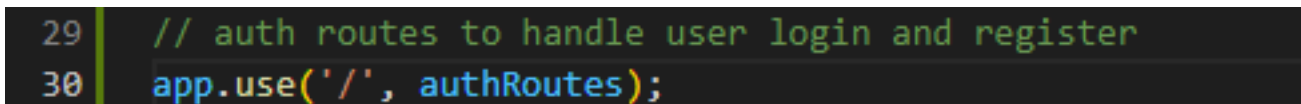
Step 5: Create authRoutes.js in routes folder



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like Config.js, base.js, controllers, Controller.js, Controller.js, ewares, js, estValidator.js, s, tModel.js, sStore.js, Model.js, and modules. The code editor shows the content of authRoutes.js in the routes folder. The code is as follows:

```
1  const express = require('express');
2  const router = express.Router();
3  const authController = require("../controllers/authController");
4
5  //Route for the user to signin
6  router.post("/login", authController.login);
7
8  //Route for the user to signup
9  router.post("/register", authController.register);
10
11 module.exports = router;
```

Step 6: Import this route in index.js file



The screenshot shows a code editor with the content of index.js. The code is as follows:

```
29 // auth routes to handle user login and register
30 app.use('/', authRoutes);
```

Step 7: Testing the newly created APIs:

POST

/register

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/register
- Body:** A JSON object with the following fields:

```
{  "username": "kumar",  "email": "kumar@test.com",  "password": "kumar123"}
```
- Response:** A 200 OK status with a response time of 125 ms and a body size of 278 B. The response body is a JSON object:

```
{  "message": "User registered successfully!"}
```

POST

/login

The screenshot shows a REST client interface with the following details:

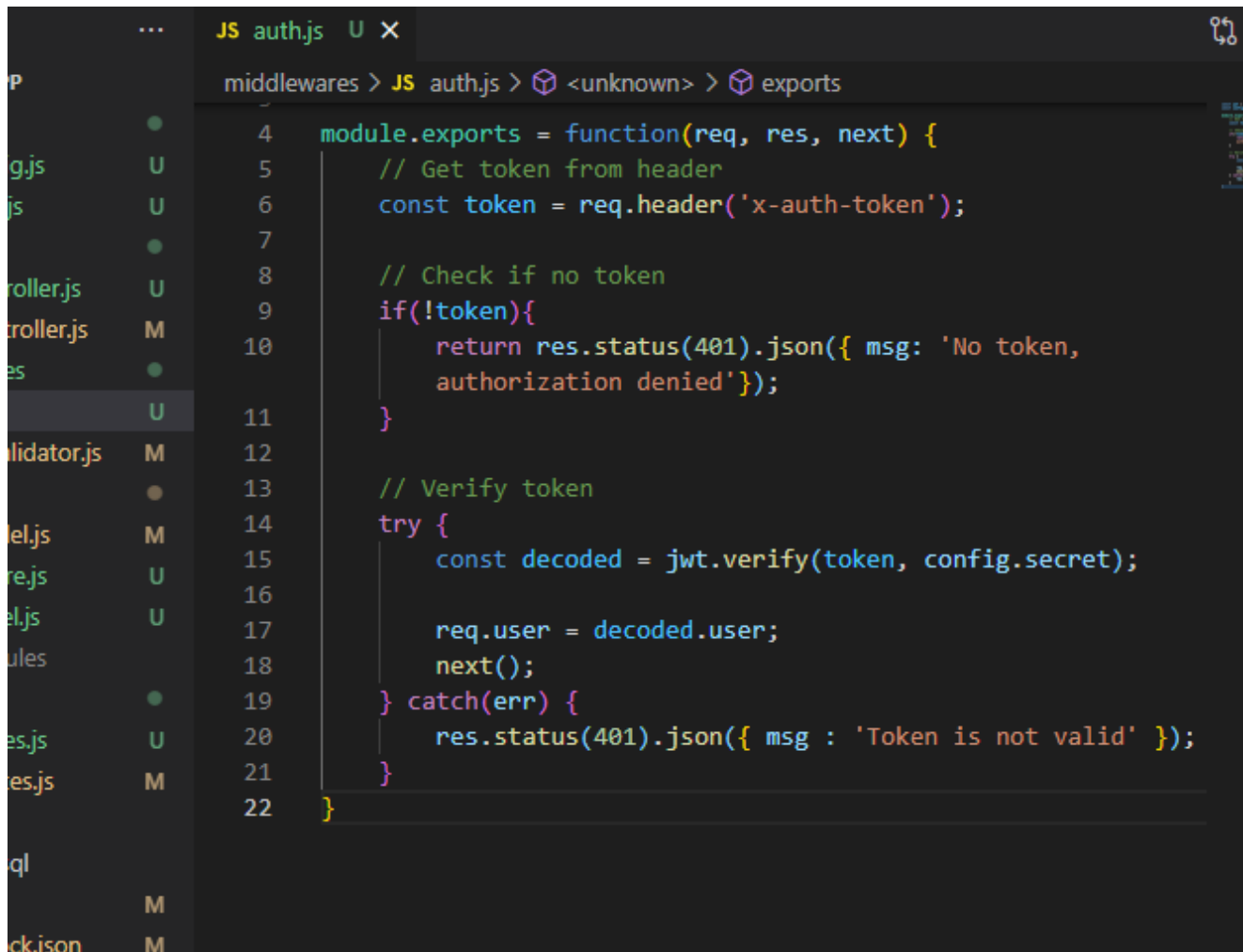
- Method:** POST
- URL:** http://localhost:8080/login
- Body:** A JSON object with the following fields:

```
{  "username": "kumar",  "password": "kumar123"}
```
- Response:** A 200 OK status with a response time of 59 ms and a body size of 442 B. The response body is a JSON object:

```
{  "id": 2,  "username": "kumar",  "email": "kumar@test.com",  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiYWV0IjoxNjQ1NTU3NDMzLCJleHAiOjE2NDU2NDM4MzN9.5CTwmA0_lryF1pM_q3Y0rfBszaYQe68_fgJfcrHcbww"}
```


2. Next, create one middleware to validate token given after login on routes such as put, post and delete books.

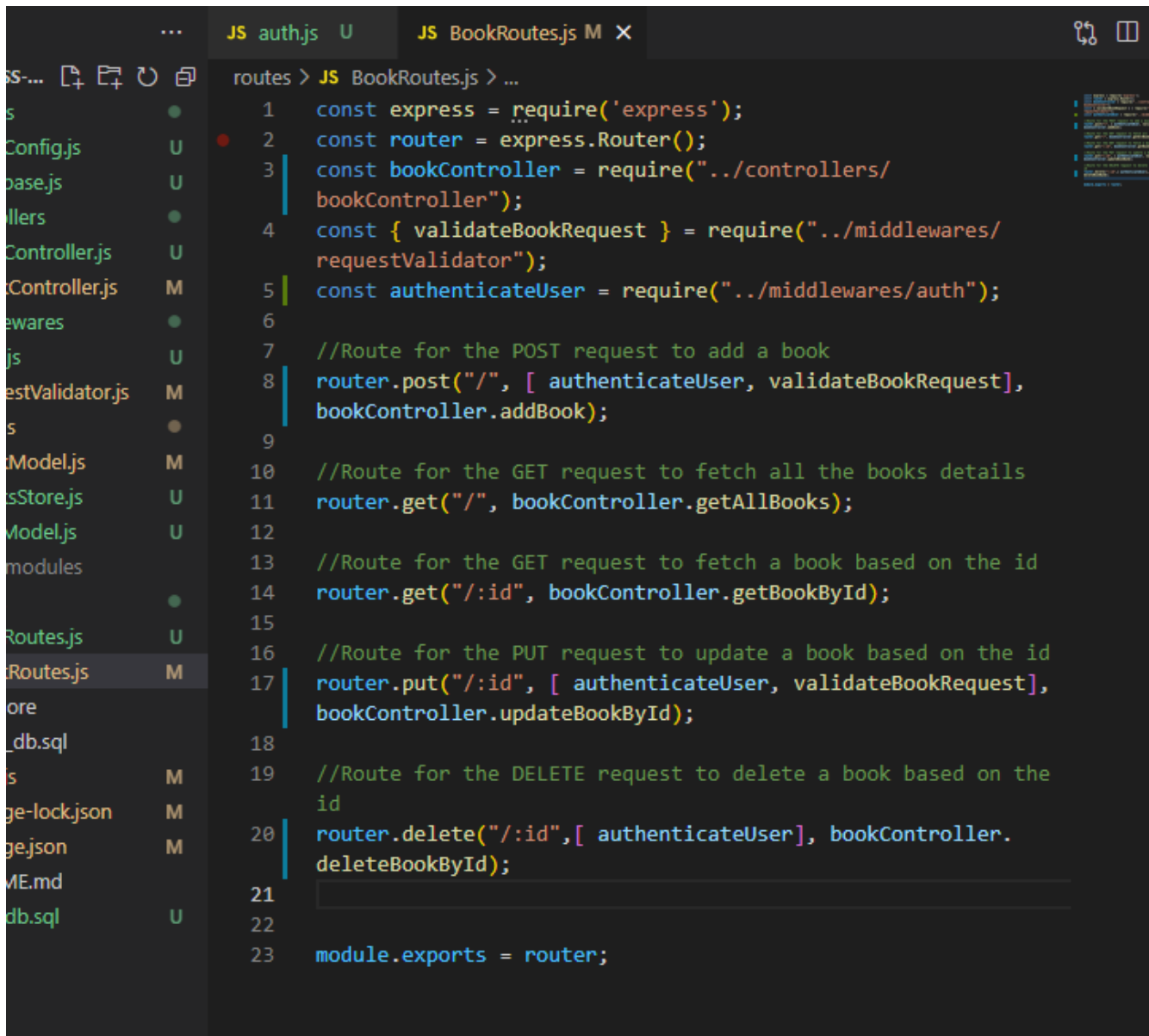
Step 1: First create an auth.js middleware inside middlewares folder



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a list of files: `auth.js`, `controller.js`, `middleware.js`, `validator.js`, `index.js`, `routes.js`, `utils.js`, `config.js`, `constants.js`, `error.js`, `index.html`, `package.json`, and `server.js`. The `auth.js` file is selected. The code editor shows the following code:

```
module.exports = function(req, res, next) {  
  // Get token from header  
  const token = req.header('x-auth-token');  
  
  // Check if no token  
  if(!token){  
    return res.status(401).json({ msg: 'No token,  
    authorization denied'});  
  }  
  
  // Verify token  
  try {  
    const decoded = jwt.verify(token, config.secret);  
  
    req.user = decoded.user;  
    next();  
  } catch(err) {  
    res.status(401).json({ msg : 'Token is not valid' });  
  }  
}
```

Step 2: Add this middlewares in bookRoutes.js file



```
... JS auth.js U JS BookRoutes.js M X
routes > JS BookRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const bookController = require("../controllers/
   bookController");
4  const { validateBookRequest } = require("../middlewares/
   requestValidator");
5  const authenticateUser = require("../middlewares/auth");
6
7  //Route for the POST request to add a book
8  router.post("/", [ authenticateUser, validateBookRequest],
   bookController.addBook);
9
10 //Route for the GET request to fetch all the books details
11 router.get("/", bookController.getAllBooks);
12
13 //Route for the GET request to fetch a book based on the id
14 router.get("/:id", bookController.getBookById);
15
16 //Route for the PUT request to update a book based on the id
17 router.put("/:id", [ authenticateUser, validateBookRequest],
   bookController.updateBookById);
18
19 //Route for the DELETE request to delete a book based on the
   id
20 router.delete("/:id",[ authenticateUser], bookController.
   deleteBookById);
21
22
23 module.exports = router;
```

Step 3: Testing api with token in header with key value as x-auth-token

PUT

/books/1

The screenshot shows the Postman interface for a PUT request to `http://localhost:8080/books/1`. The request is configured with the following headers:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e...	
Key	Value	Description

The response is displayed in the 'Body' tab, showing a JSON message:

```
1 {
2   "message": "Book Id: 1 Updated"
3 }
```

The status bar indicates a 200 OK response with a response time of 66 ms and a body size of 267 B.

Please find the github code link [here](#).