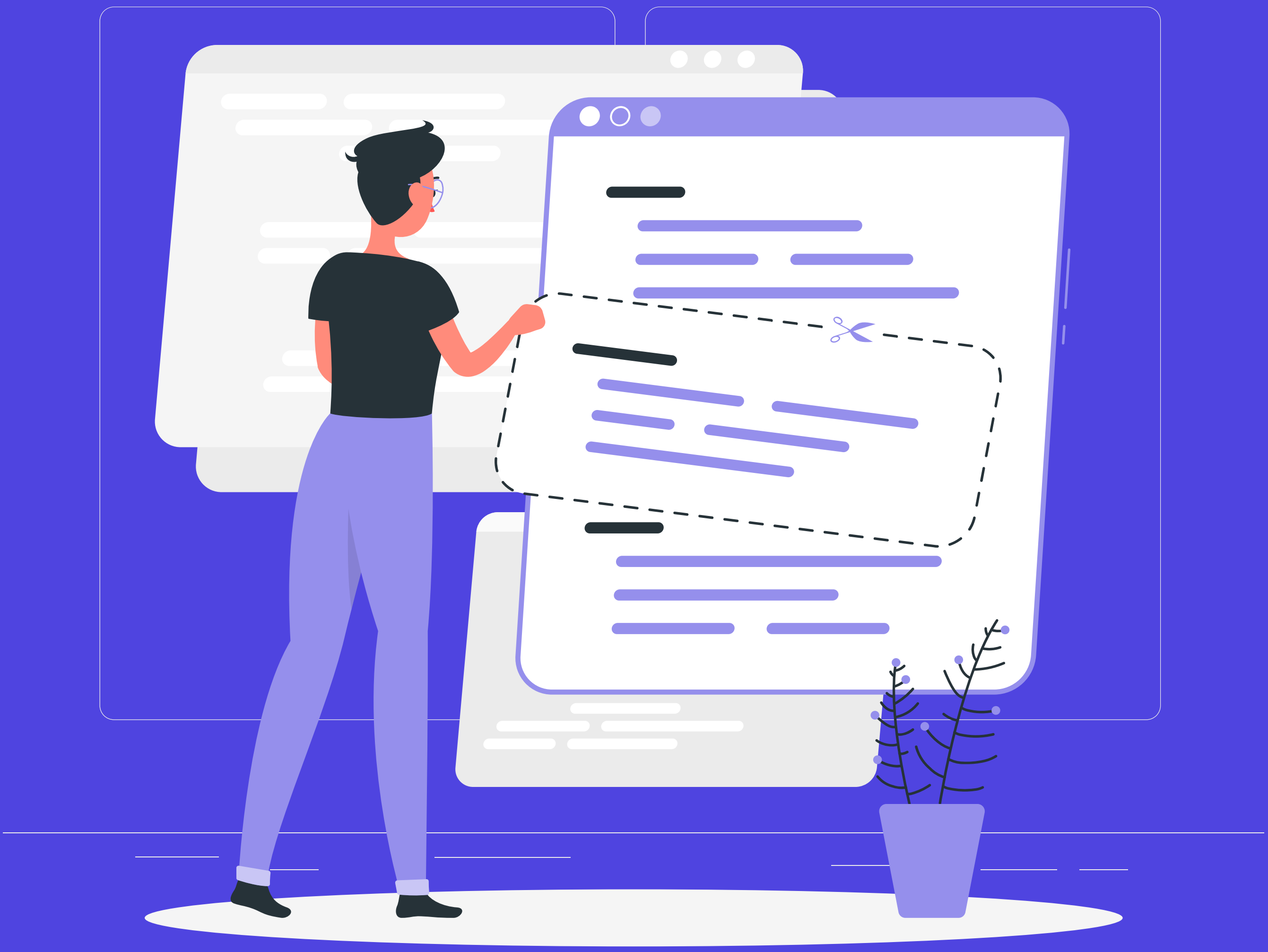
Creating Authenticated Cart APIs
Assignment Solution

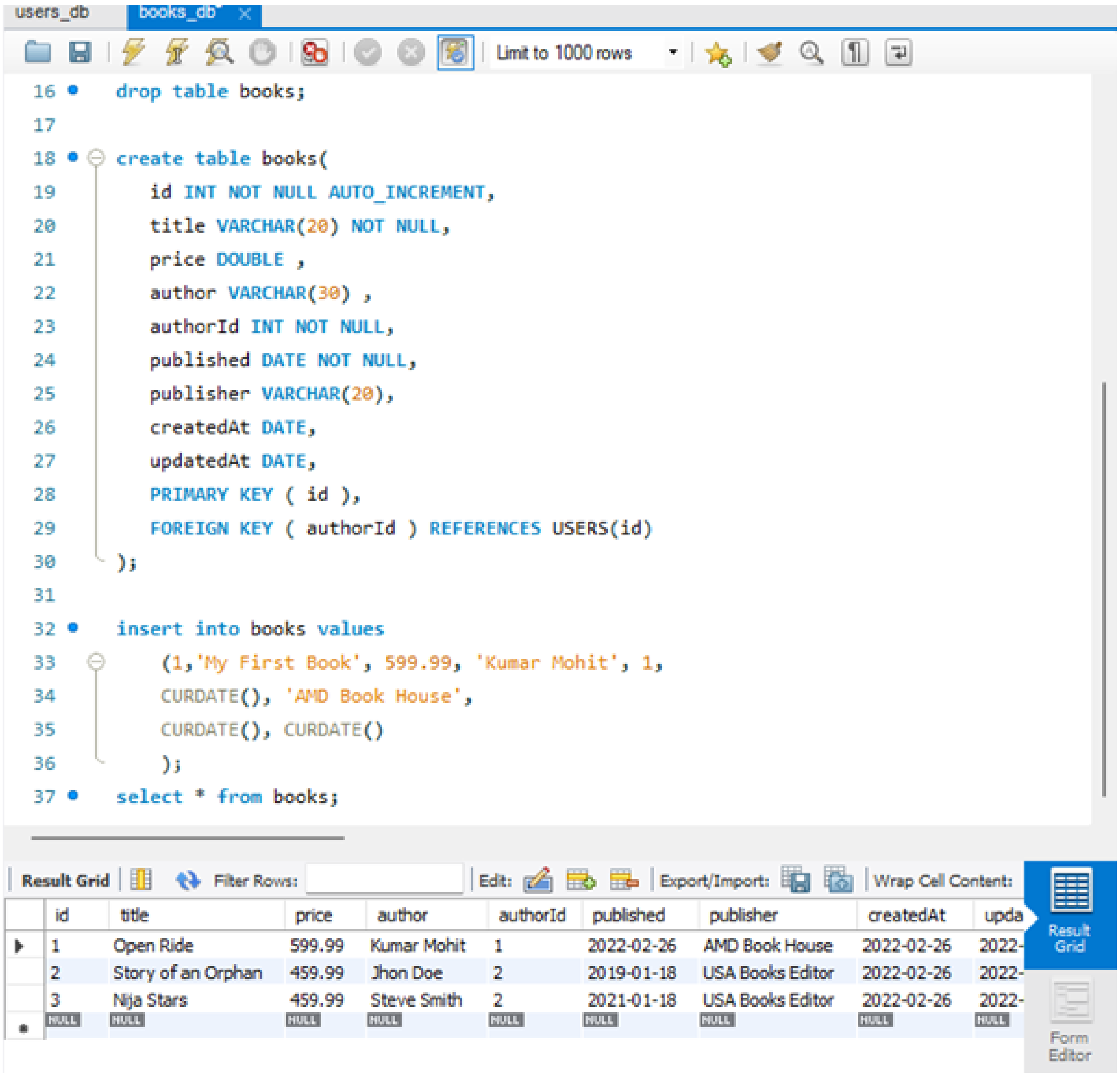


Relevel
by Unacademy

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

1. Modify table books by adding another column named user with value of user id to make only owner of a book to modify or delete it.

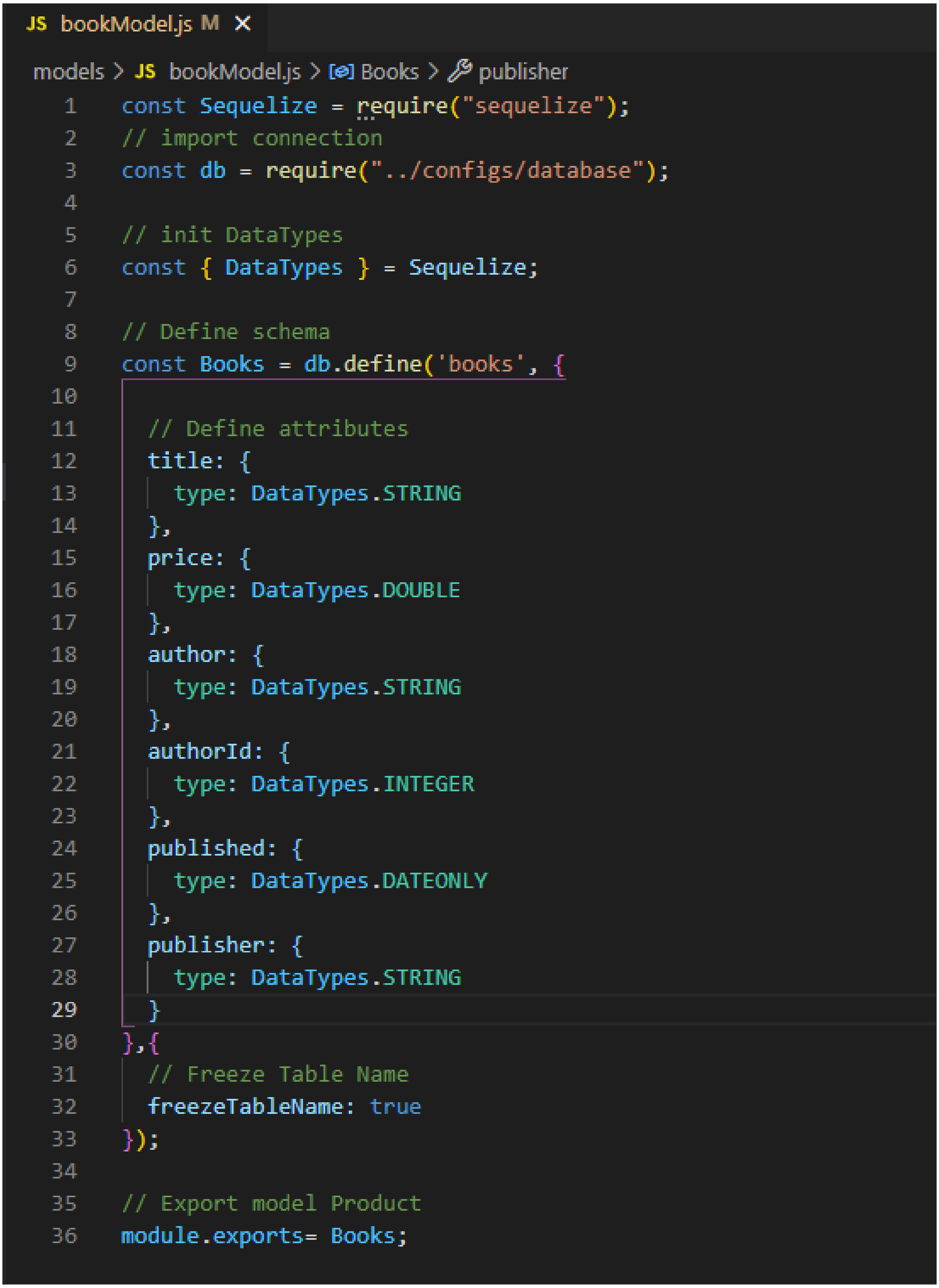
Step 1: First let us add another column authorId in books table and add the reference to it using foreign key relation:



```
drop table books;

create table books(
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(20) NOT NULL,
    price DOUBLE ,
    author VARCHAR(30) ,
    authorId INT NOT NULL,
    published DATE NOT NULL,
    publisher VARCHAR(20),
    createdAt DATE,
    updatedAt DATE,
    PRIMARY KEY ( id ),
    FOREIGN KEY ( authorId ) REFERENCES USERS(id)
);

insert into books values
     (1,'My First Book', 599.99, 'Kumar Mohit', 1,
     CURDATE(), 'AMD Book House',
     CURDATE(), CURDATE()
     );
select * from books;
```

id	title	price	author	authorId	published	publisher	createdAt	upda
1	Open Ride	599.99	Kumar Mohit	1	2022-02-26	AMD Book House	2022-02-26	2022-
2	Story of an Orphan	459.99	Jhon Doe	2	2019-01-18	USA Books Editor	2022-02-26	2022-
3	Nija Stars	459.99	Steve Smith	2	2021-01-18	USA Books Editor	2022-02-26	2022-
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Step 2: Modify the bookModel.js accordingly:

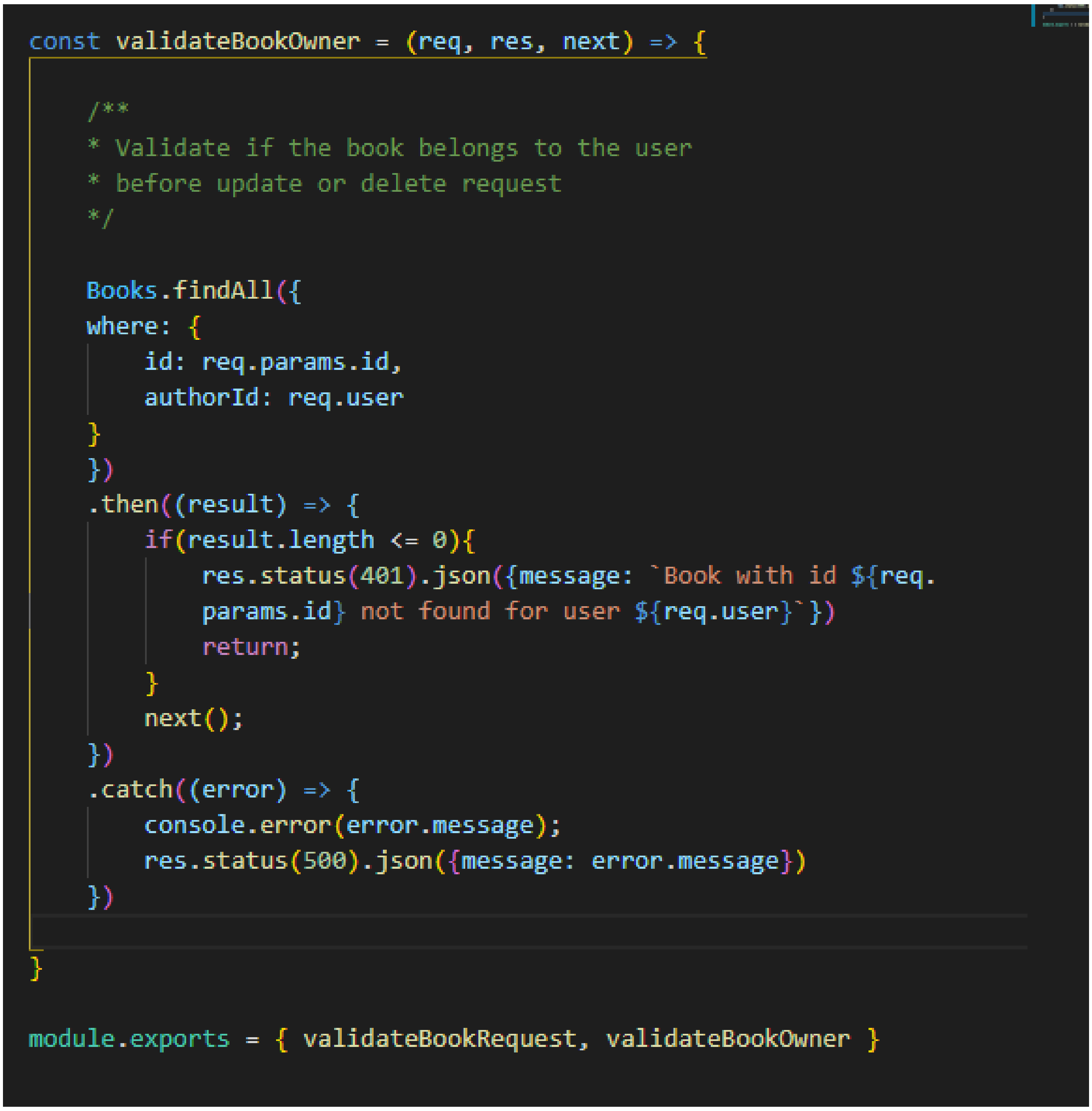


bookModel.js M ×

models > bookModel.js > Books > publisher

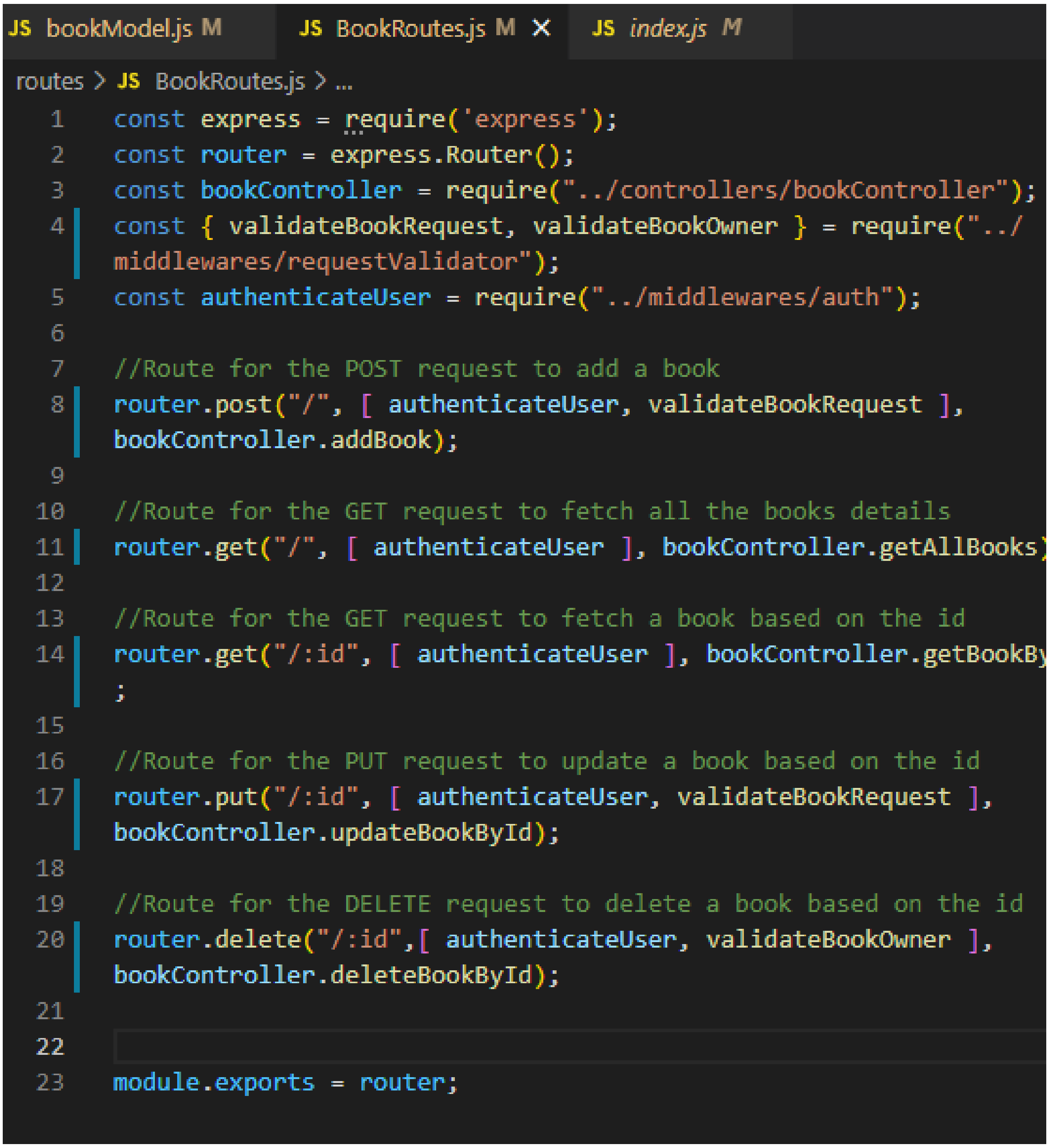
```javascript
const Sequelize = require("sequelize");
// import connection
const db = require("../configs/database");

// init DataTypes
const { DataTypes } = Sequelize;

// Define schema
const Books = db.define('books', {

  // Define attributes
  title: {
    type: DataTypes.STRING
  },
  price: {
    type: DataTypes.DOUBLE
  },
  author: {
    type: DataTypes.STRING
  },
  authorId: {
    type: DataTypes.INTEGER
  },
  published: {
    type: DataTypes.DATEONLY
  },
  publisher: {
    type: DataTypes.STRING
  }
},{
  // Freeze Table Name
  freezeTableName: true
});

// Export model Product
module.exports= Books;
```

Step 3: Now let us add a middleware to verify book owner:



```
const validateBookOwner = (req, res, next) => {

    /**
    * Validate if the book belongs to the user
    * before update or delete request
    */

    Books.findAll({
    where: {
        id: req.params.id,
        authorId: req.user
    }
    })
    .then((result) => {
        if(result.length <= 0){
            res.status(401).json({message: `Book with id ${req.
            params.id} not found for user ${req.user}`})
            return;
        }
        next();
    })
    .catch((error) => {
        console.error(error.message);
        res.status(500).json({message: error.message})
    })

}

module.exports = { validateBookRequest, validateBookOwner }
```

Step 4: Now add this middleware before the routes to validate the request:

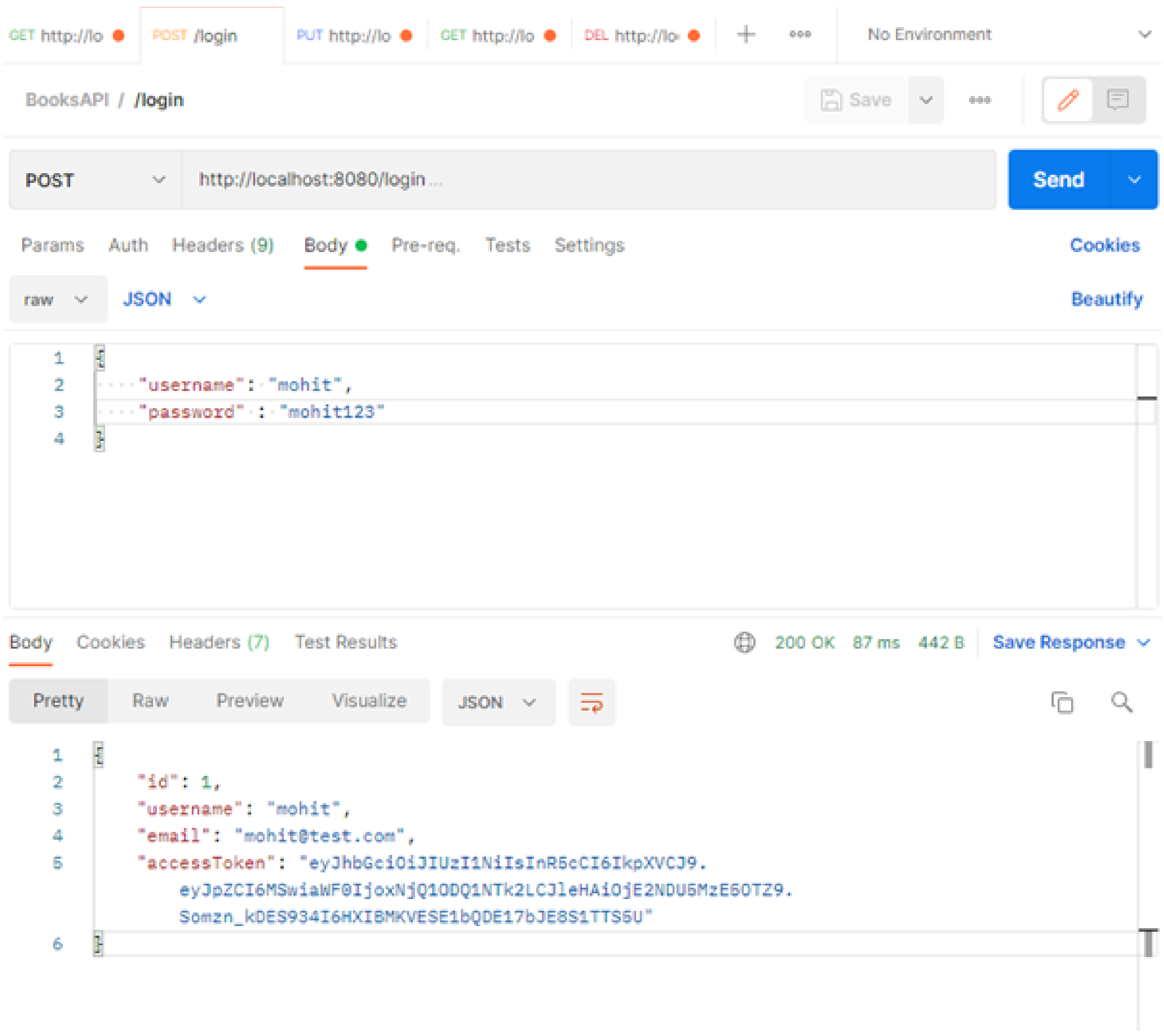


JS bookModel.js M | JS BookRoutes.js M ✕ | JS index.js M

routes > JS BookRoutes.js > ...

```
const express = require('express');
const router = express.Router();
const bookController = require("../controllers/bookController");
const { validateBookRequest, validateBookOwner } = require("../
middlewares/requestValidator");
const authenticateUser = require("../middlewares/auth");

//Route for the POST request to add a book
router.post("/", [ authenticateUser, validateBookRequest ],
bookController.addBook);

//Route for the GET request to fetch all the books details
router.get("/", [ authenticateUser ], bookController.getAllBooks
//Route for the GET request to fetch a book based on the id
router.get("/:id", [ authenticateUser ], bookController.getBookBy
;

//Route for the PUT request to update a book based on the id
router.put("/:id", [ authenticateUser, validateBookRequest ],
bookController.updateBookById);

//Route for the DELETE request to delete a book based on the id
router.delete("/:id",[ authenticateUser, validateBookOwner ],
bookController.deleteBookById);


module.exports = router;
```

Step 5: Testing APIs:
POST /login

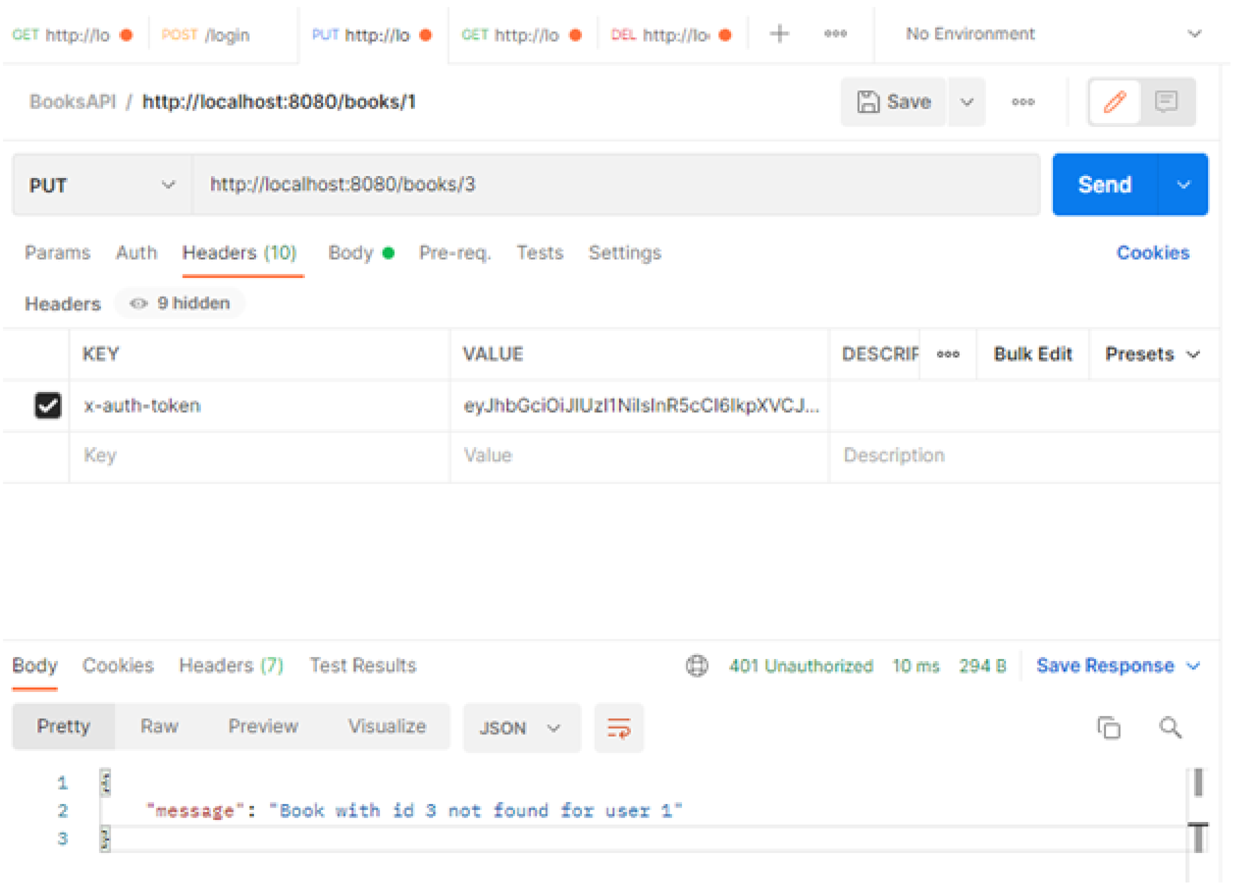


POST http://localhost:8080/login

```
{
    "username": "mohit",
    "password" : "mohit123"
}
```

```
{
    "id": 1,
    "username": "mohit",
    "email": "mohit@test.com",
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6MSwiaWF0IjoxNjQ1ODQ1NTk2LCJleHAiOjE2NDU5MzE5OTZ9.
        Somzn_kDES934I6HXIBMKVESE1bQDE17bJE8S1TTS6U"
}
```

PUT /books/:id



BooksAPI / http://localhost:8080/books/1

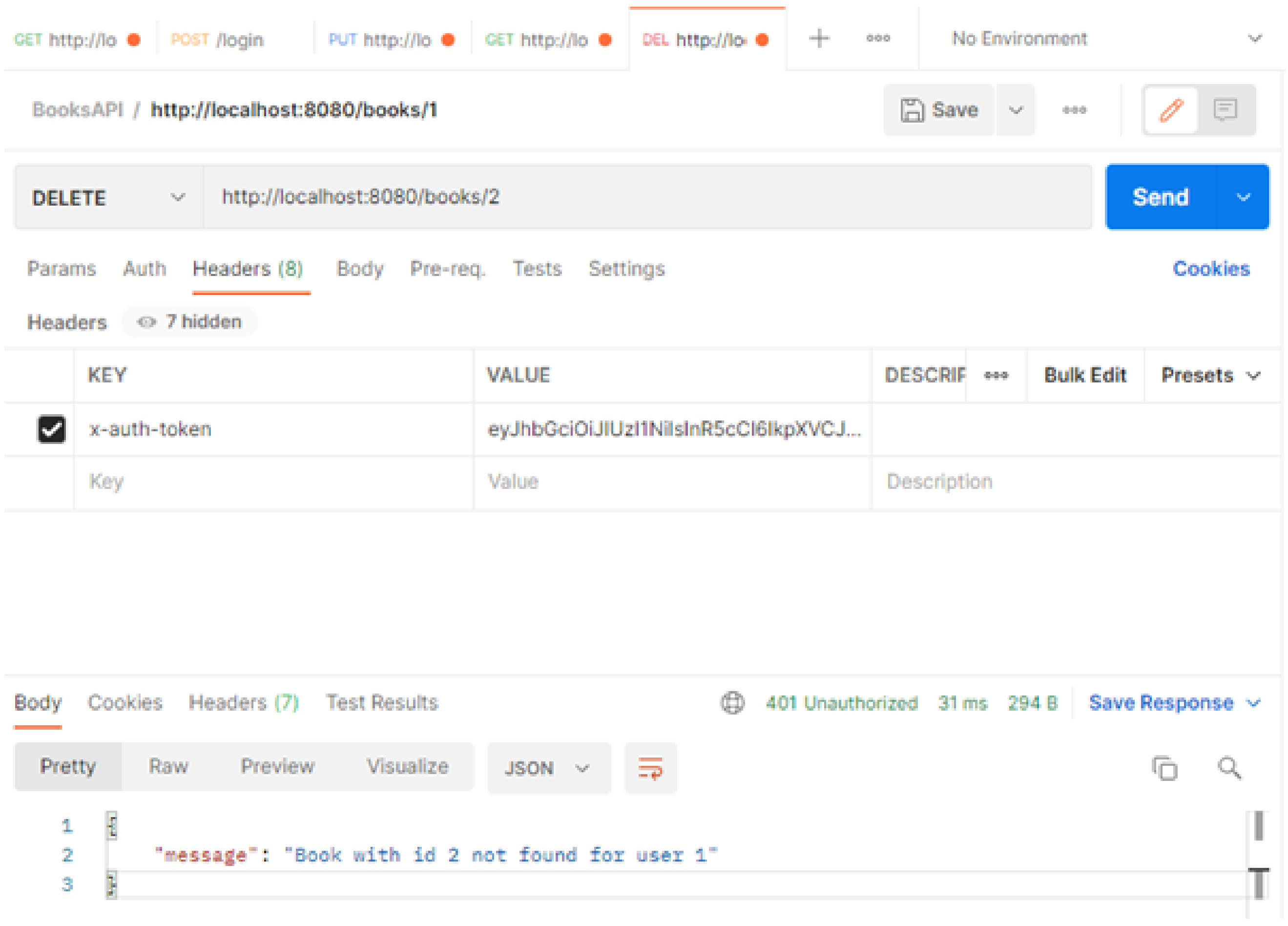
PUT http://localhost:8080/books/3

	KEY	VALUE	DESCRIPTION
✓	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	

401 Unauthorized 10 ms 294 B

```
{
    "message": "Book with id 3 not found for user 1"
}
```

DELETE /books/:id



BooksAPI / http://localhost:8080/books/1

DELETE http://localhost:8080/books/2

	KEY	VALUE	DESCRIPTION
✓	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	

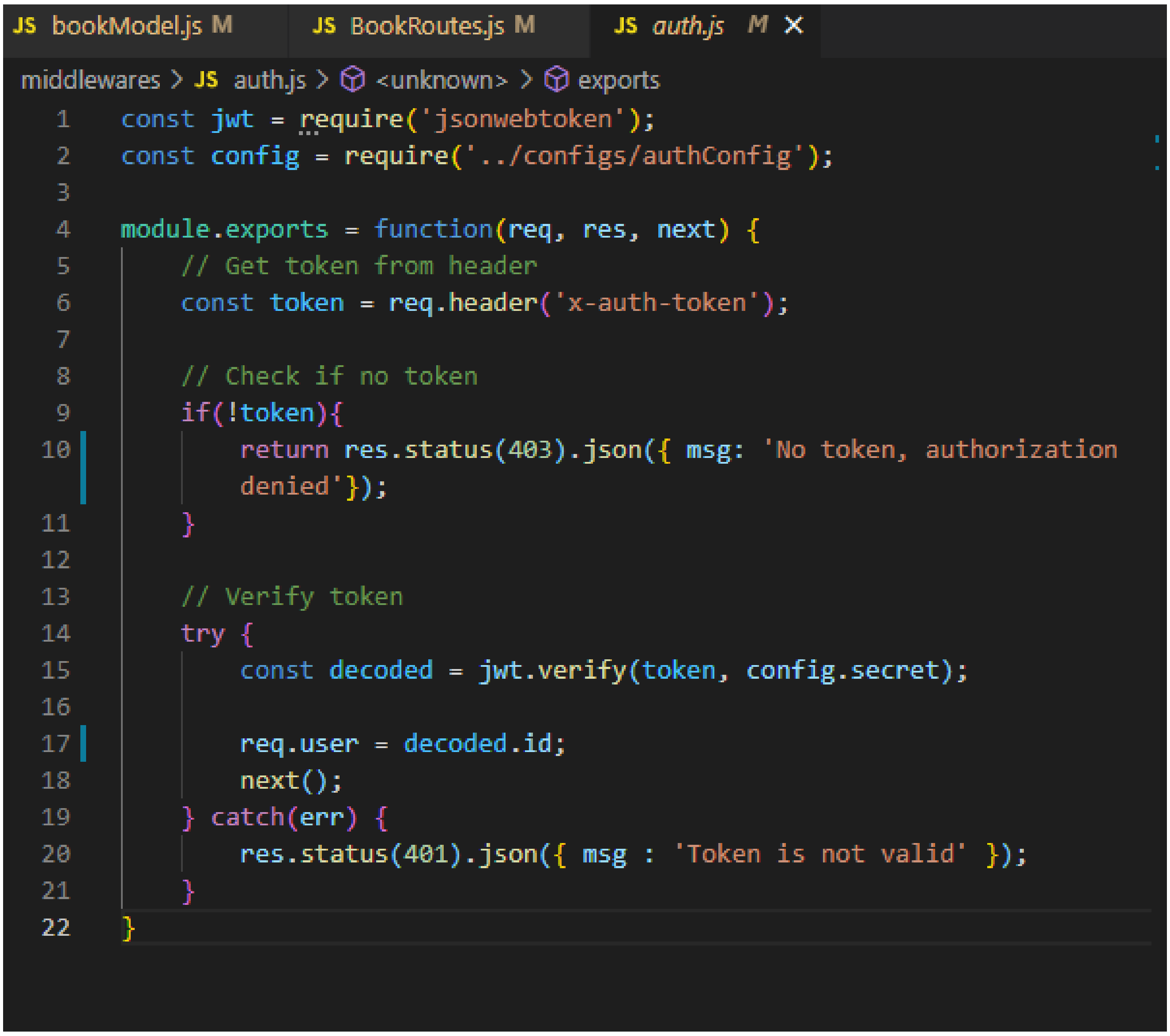
401 Unauthorized 31 ms 294 B

```
{
    "message": "Book with id 2 not found for user 1"
}
```

Relevel
by Unacademy

2. Similarly modify get all books routes to only fetch logged in user books only.

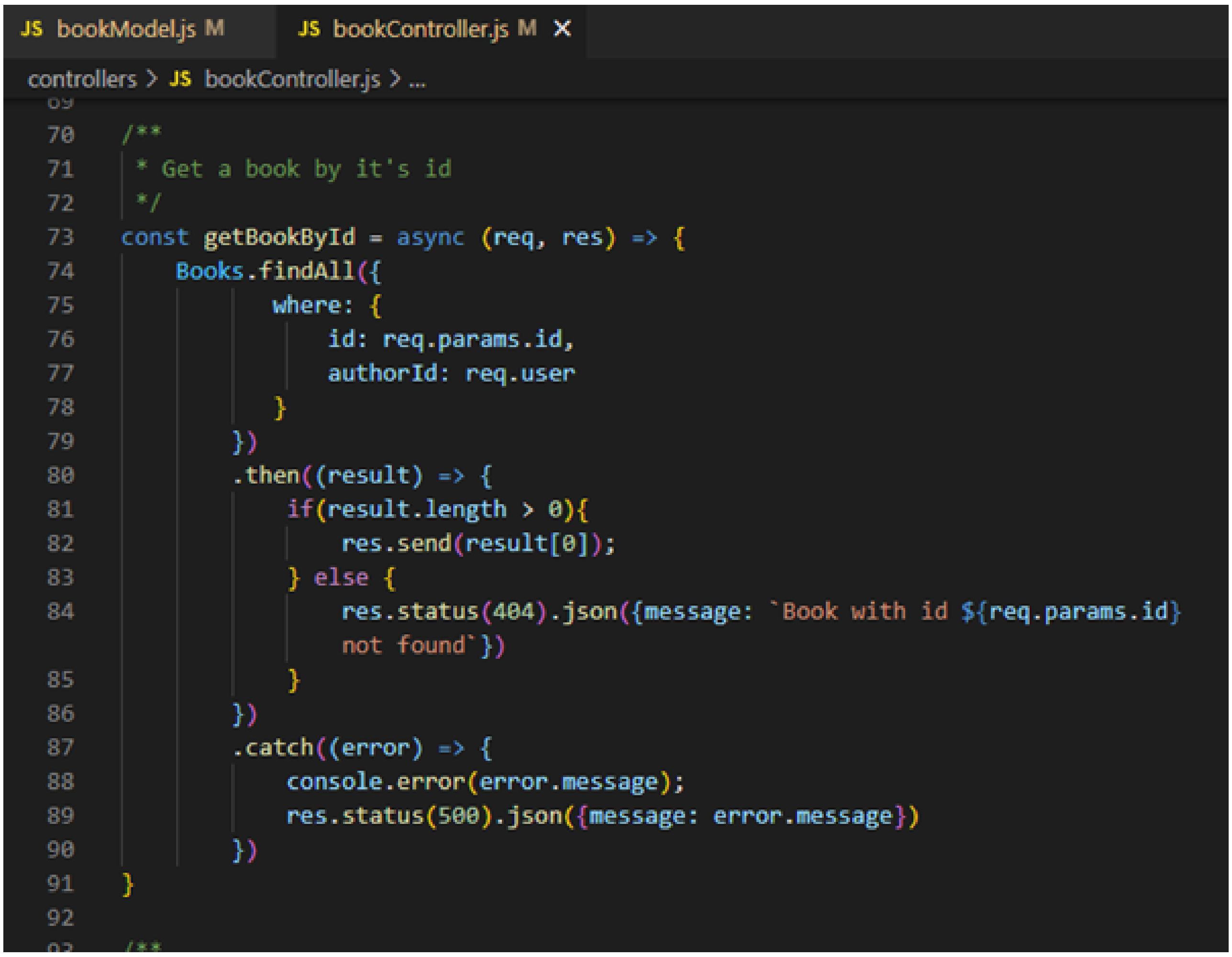
As shown above in routes file we have added authentication before /books and /books/:id as well.
This will not only secure the request but also add user id in every request to use it in different controllers:



JS bookModel.js M   JS BookRoutes.js M   JS auth.js M ×

middlewares > JS auth.js > <unknown> > exports

```javascript
const jwt = require('jsonwebtoken');
const config = require('../configs/authConfig');

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(403).json({ msg: 'No token, authorization denied'});
    }

    // Verify token
    try {
        const decoded = jwt.verify(token, config.secret);

        req.user = decoded.id;
        next();
    } catch(err) {
        res.status(401).json({ msg : 'Token is not valid' });
    }
}
```

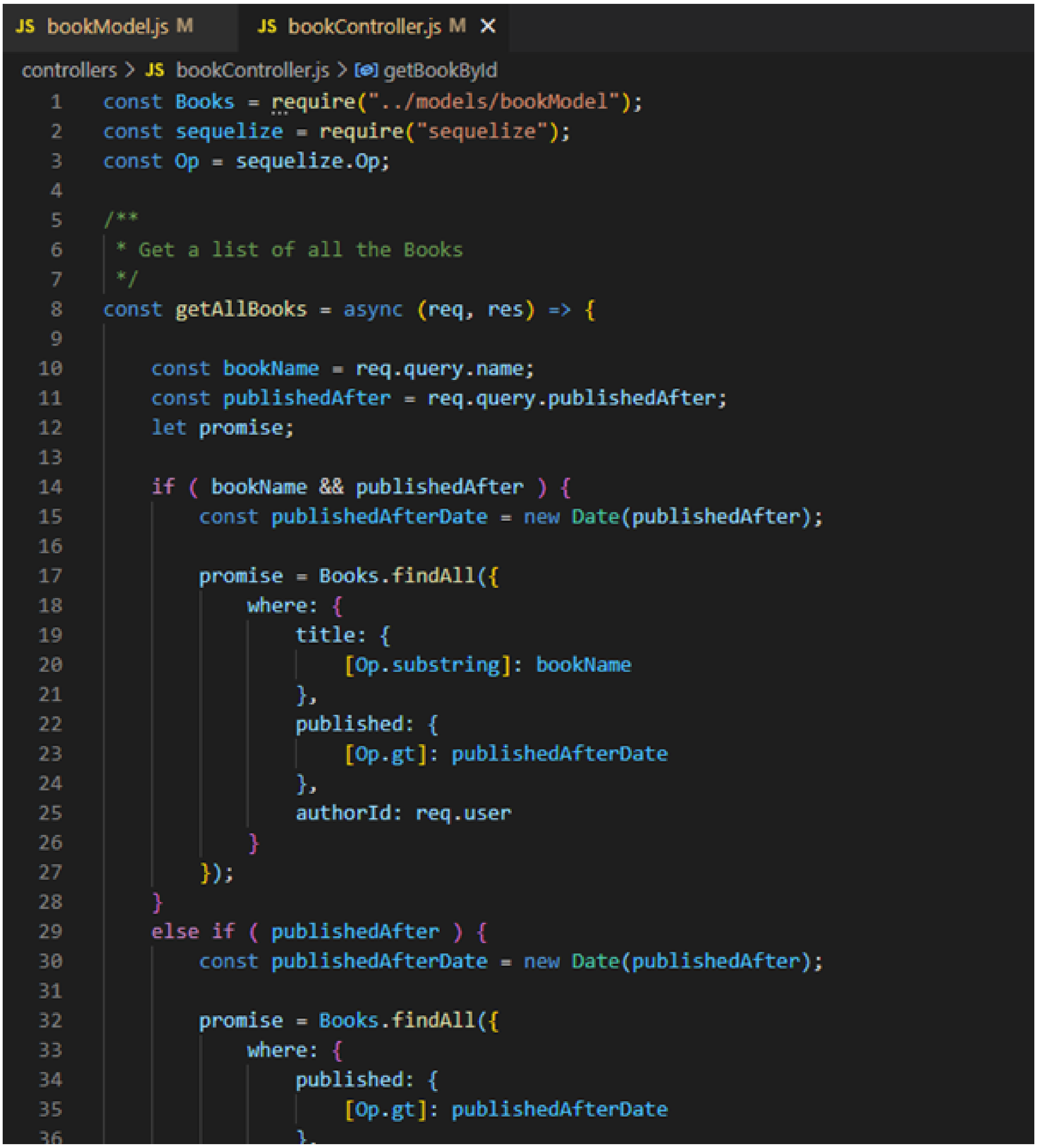

JS bookModel.js M   JS bookController.js M ✕

controllers > JS bookController.js > ...

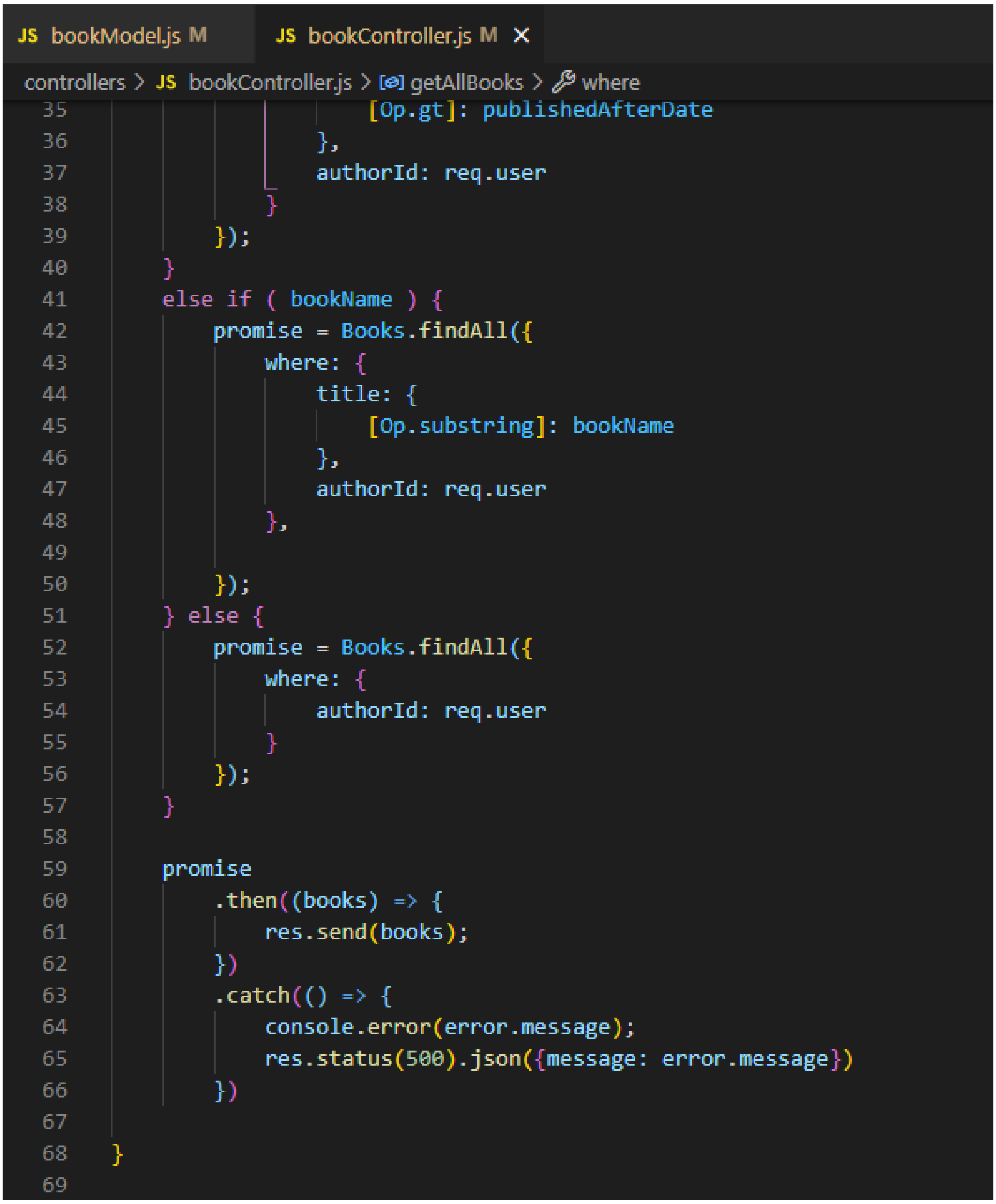
```
/**
 * Get a book by it's id
 */
const getBookById = async (req, res) => {
    Books.findAll({
        where: {
            id: req.params.id,
            authorId: req.user
        }
    })
    .then((result) => {
        if(result.length > 0){
            res.send(result[0]);
        } else {
            res.status(404).json({message: `Book with id ${req.params.id}
            not found`})
        }
    })
    .catch((error) => {
        console.error(error.message);
        res.status(500).json({message: error.message})
    })
}

/**
```

As shown above we have now added another where clause to look fetch the book with authorId only.

Similarly, in /books request controller:



```
const Books = require("../models/bookModel");
const sequelize = require("sequelize");
const Op = sequelize.Op;

/**
 * Get a list of all the Books
 */
const getAllBooks = async (req, res) => {

    const bookName = req.query.name;
    const publishedAfter = req.query.publishedAfter;
    let promise;

    if ( bookName && publishedAfter ) {
        const publishedAfterDate = new Date(publishedAfter);

        promise = Books.findAll({
            where: {
                title: {
                    [Op.substring]: bookName
                },
                published: {
                    [Op.gt]: publishedAfterDate
                },
                authorId: req.user
            }
        });
    }
    else if ( publishedAfter ) {
        const publishedAfterDate = new Date(publishedAfter);

        promise = Books.findAll({
            where: {
                published: {
                    [Op.gt]: publishedAfterDate
```



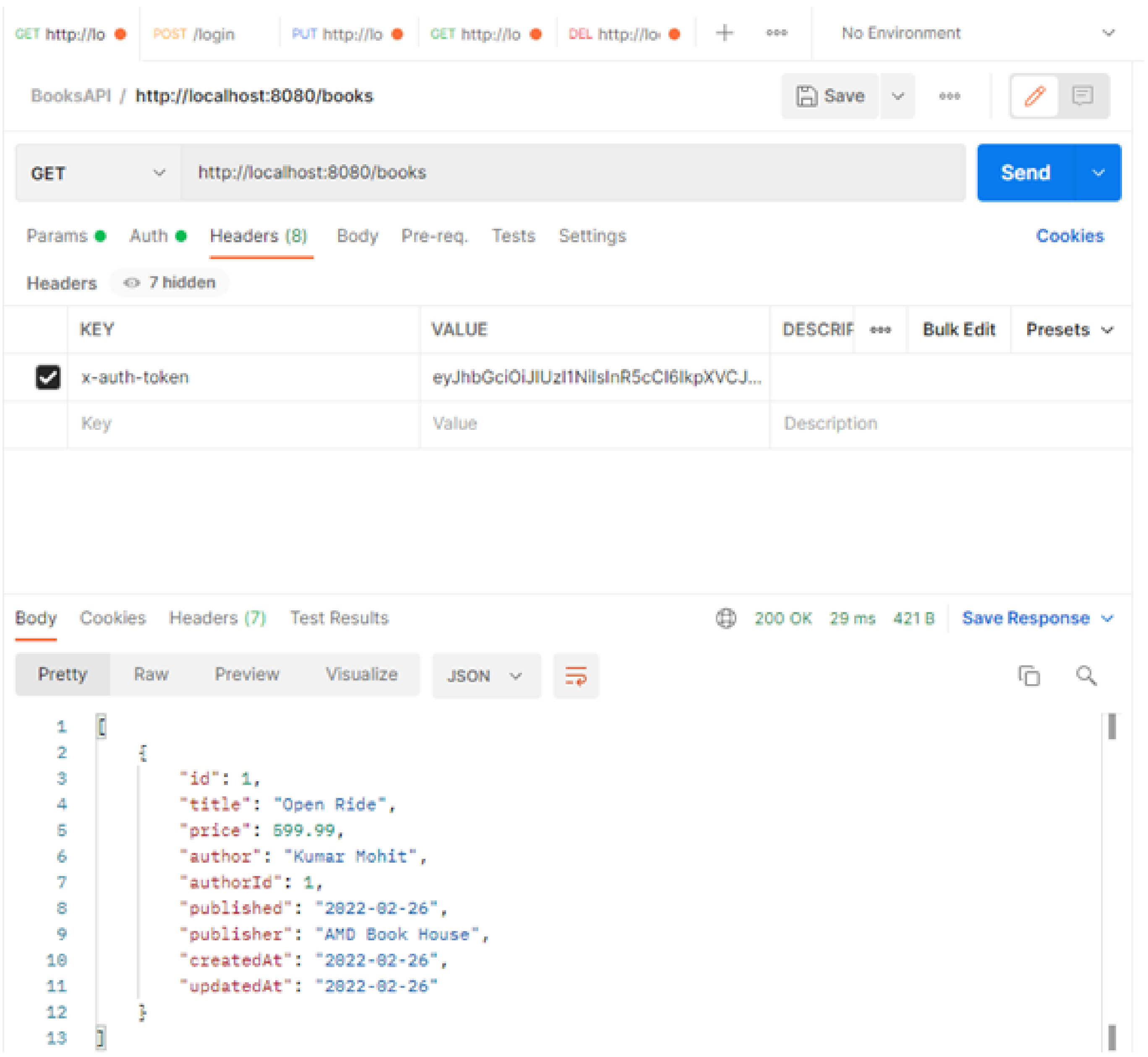
bookModel.js M   bookController.js M ×

controllers > JS bookController.js > getAllBooks > where

```
                    [Op.gt]: publishedAfterDate
                },
                authorId: req.user
            }
        });
    }
    else if ( bookName ) {
        promise = Books.findAll({
            where: {
                title: {
                    [Op.substring]: bookName
                },
                authorId: req.user
            },

        });
    } else {
        promise = Books.findAll({
            where: {
                authorId: req.user
            }
        });
    }

    promise
        .then((books) => {
            res.send(books);
        })
        .catch(() => {
            console.error(error.message);
            res.status(500).json({message: error.message})
        })

}

```

Testing APIs:

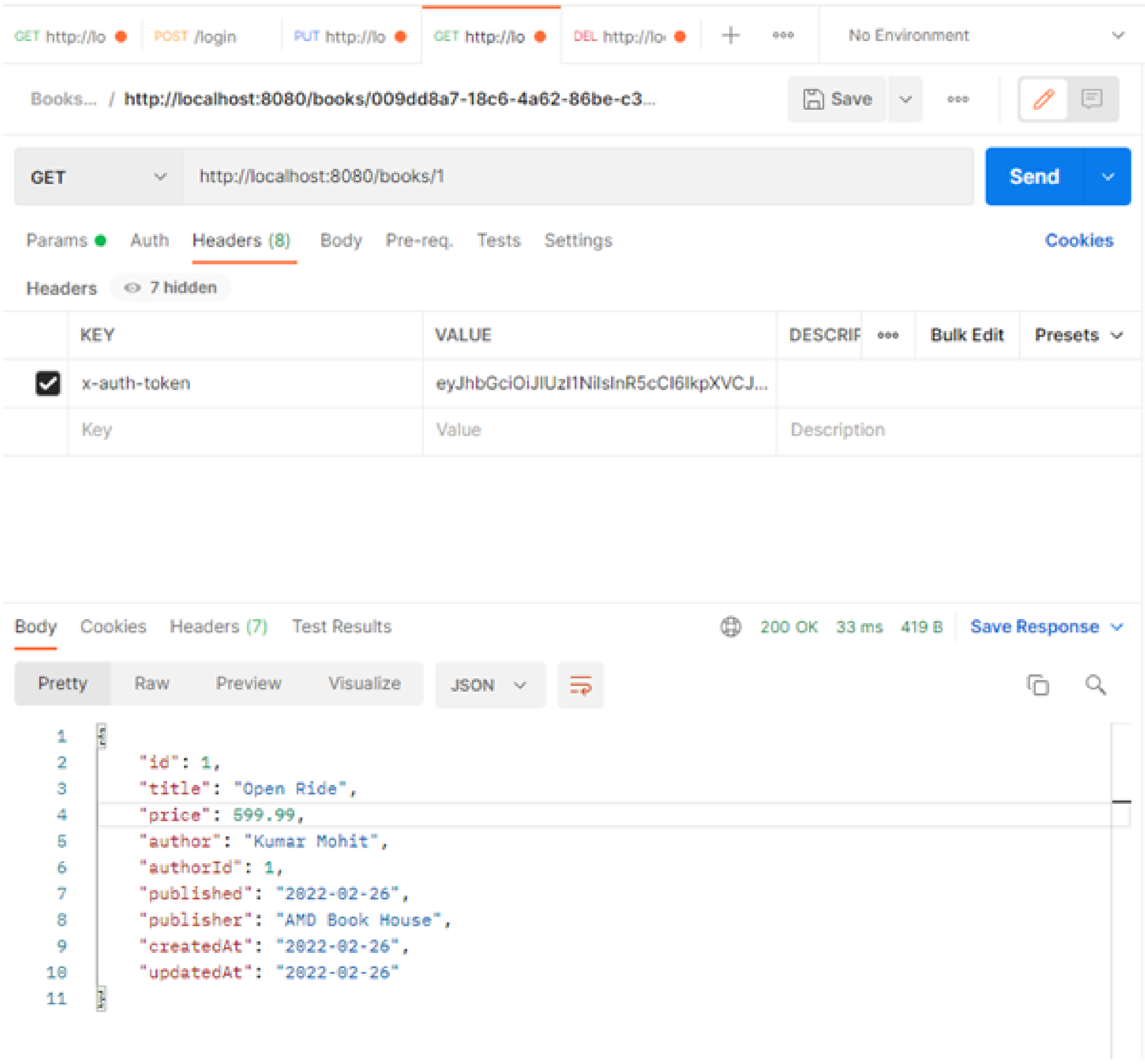
GET /books



KEY	VALUE	DESCRIP
✓ x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	
Key	Value	Description

```
[
    {
        "id": 1,
        "title": "Open Ride",
        "price": 599.99,
        "author": "Kumar Mohit",
        "authorId": 1,
        "published": "2022-02-26",
        "publisher": "AMD Book House",
        "createdAt": "2022-02-26",
        "updatedAt": "2022-02-26"
    }
]
```

GET /books/:id



KEY	VALUE	DESCRIPTION
✓ x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	
Key	Value	Description

```
{
    "id": 1,
    "title": "Open Ride",
    "price": 599.99,
    "author": "Kumar Mohit",
    "authorId": 1,
    "published": "2022-02-26",
    "publisher": "AMD Book House",
    "createdAt": "2022-02-26",
    "updatedAt": "2022-02-26"
}
```

Please find the code github link for your reference.