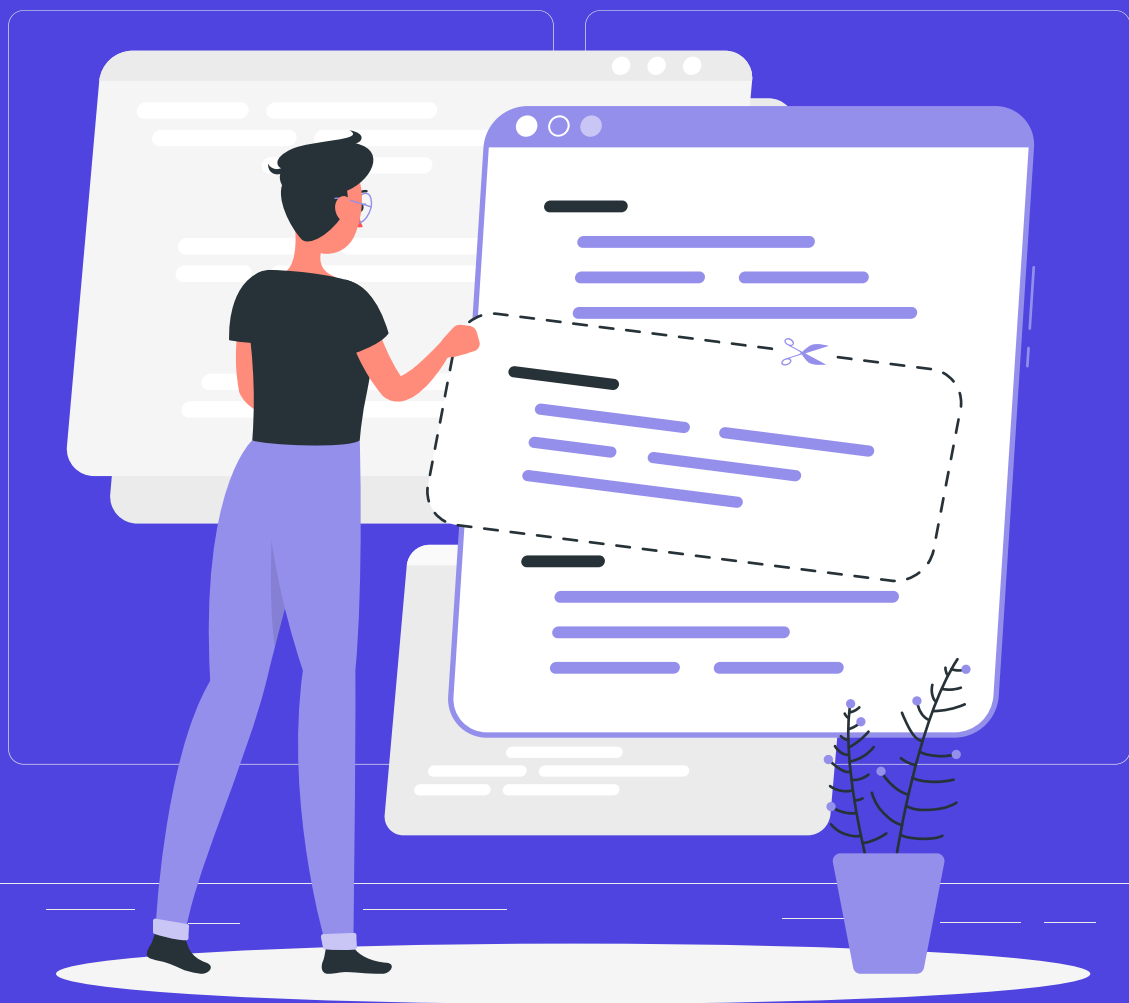


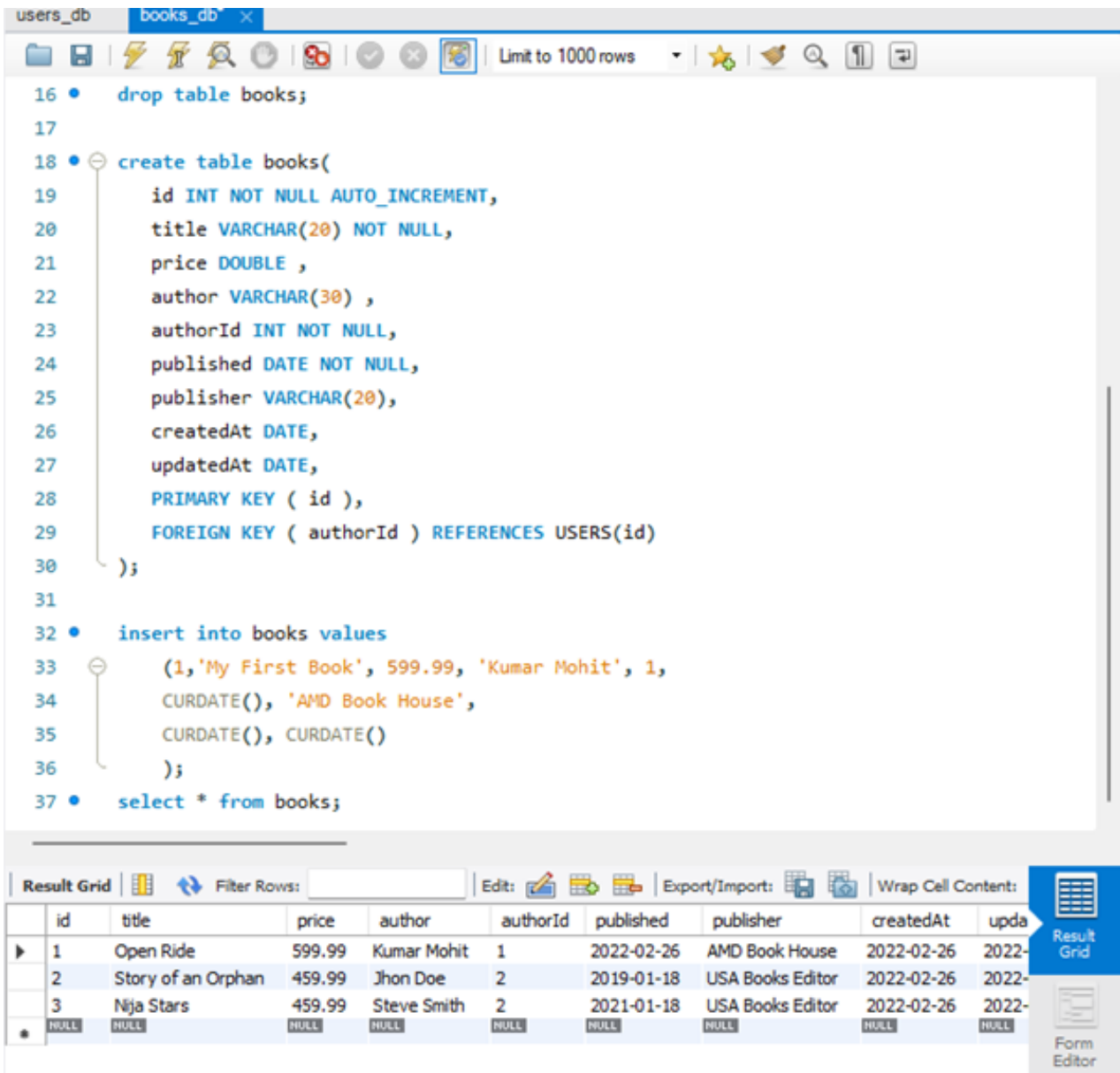
Creating Authenticated Cart APIs Assignment Solution



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:



The screenshot shows a database management tool with two tabs: 'users_db' and 'books_db'. The 'books_db' tab is active, displaying a SQL editor with the following queries:

```
16 • drop table books;
17
18 • create table books(
19     id INT NOT NULL AUTO_INCREMENT,
20     title VARCHAR(20) NOT NULL,
21     price DOUBLE ,
22     author VARCHAR(30) ,
23     authorId INT NOT NULL,
24     published DATE NOT NULL,
25     publisher VARCHAR(20),
26     createdAt DATE,
27     updatedAt DATE,
28     PRIMARY KEY ( id ),
29     FOREIGN KEY ( authorId ) REFERENCES USERS(id)
30 );
31
32 • insert into books values
33     (1,'My First Book', 599.99, 'Kumar Mohit', 1,
34     CURDATE(), 'AMD Book House',
35     CURDATE(), CURDATE())
36 );
37 • select * from books;
```

Below the SQL editor is a 'Result Grid' showing the data from the 'books' table. The grid has columns: id, title, price, author, authorId, published, publisher, createdAt, and updatedAt. The data is as follows:

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

On the right side of the result grid, there are two buttons: 'Result Grid' and 'Form Editor'.

Step 2: Modify the bookModel.js accordingly:

JS bookModel.js M X

```
models > JS bookModel.js > Books > publisher
1  const Sequelize = require("sequelize");
2  // import connection
3  const db = require("../configs/database");
4
5  // init DataTypes
6  const { DataTypes } = Sequelize;
7
8  // Define schema
9  const Books = db.define('books', {
10
11    // Define attributes
12    title: {
13      type: DataTypes.STRING
14    },
15    price: {
16      type: DataTypes.DOUBLE
17    },
18    author: {
19      type: DataTypes.STRING
20    },
21    authorId: {
22      type: DataTypes.INTEGER
23    },
24    published: {
25      type: DataTypes.DATEONLY
26    },
27    publisher: {
28      type: DataTypes.STRING
29    }
30  }, {
31    // Freeze Table Name
32    freezeTableName: true
33  });
34
35  // Export model Product
36  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 X    JS index.js M
routes > JS BookRoutes.js > ...
 1  const express = require('express');
 2  const router = express.Router();
 3  const bookController = require("../controllers/bookController");
 4  const { validateBookRequest, validateBookOwner } = 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("/", [ authenticateUser ], bookController.getAllBooks);
12
13  //Route for the GET request to fetch a book based on the id
14  router.get("/:id", [ authenticateUser ], bookController.getBookBy
    ;
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, validateBookOwner ],
    bookController.deleteBookById);
21
22
23  module.exports = router;
```

Step 5: Testing APIs:

POST /login

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

- Request Method:** POST
- URL:** http://localhost:8080/login...
- Body (JSON):**

```
{
  "username": "mohit",
  "password": "mohit123"
}
```
- Response Status:** 200 OK
- Response Time:** 87 ms
- Response Size:** 442 B
- Response Body (JSON):**

```
{
  "id": 1,
  "username": "mohit",
  "email": "mohit@test.com",
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjQ1ODQ1NTk2LCJleHAiOjE2NDU5MzE5OTZ9.Somzn_kDES934I6HXBKVESE1bQDE17bJE8S1TTS5U"
}
```

PUT /books/:id

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

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

Send

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies

Headers 9 hidden

	KEY	VALUE	DESCRIF	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...				
	Key	Value	Description			

Body Cookies Headers (7) Test Results 401 Unauthorized 10 ms 294 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2  "message": "Book with id 3 not found for user 1"
3

```

DELETE /books/:id

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

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

Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Headers 7 hidden

	KEY	VALUE	DESCRIF	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...				
	Key	Value	Description			

Body Cookies Headers (7) Test Results 401 Unauthorized 31 ms 294 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2  "message": "Book with id 2 not found for user 1"
3

```

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 X
middlewares > JS auth.js > <unknown> > exports
1  const jwt = require('jsonwebtoken');
2  const config = require('../configs/authConfig');
3
4  module.exports = function(req, res, next) {
5      // Get token from header
6      const token = req.header('x-auth-token');
7
8      // Check if no token
9      if(!token){
10         return res.status(403).json({ msg: 'No token, authorization
            denied'});
11     }
12
13     // Verify token
14     try {
15         const decoded = jwt.verify(token, config.secret);
16
17         req.user = decoded.id;
18         next();
19     } catch(err) {
20         res.status(401).json({ msg : 'Token is not valid' });
21     }
22 }
```



```
JS bookModel.js M    JS bookController.js M X
controllers > JS bookController.js > ...
69
70 /**
71  * Get a book by it's id
72  */
73 const getBookById = async (req, res) => {
74   Books.findAll({
75     where: {
76       id: req.params.id,
77       authorId: req.user
78     }
79   })
80   .then((result) => {
81     if(result.length > 0){
82       res.send(result[0]);
83     } else {
84       res.status(404).json({message: `Book with id ${req.params.id}
85         not found`})
86     }
87   })
88   .catch((error) => {
89     console.error(error.message);
90     res.status(500).json({message: error.message})
91   })
92 }
93 /**
```

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

Similarly, in /books request controller:

JS bookModel.js M

JS bookController.js M X

controllers > JS bookController.js > getBookById

```
1  const Books = require("../models/bookModel");
2  const sequelize = require("sequelize");
3  const Op = sequelize.Op;
4
5  /**
6   * Get a list of all the Books
7   */
8  const getAllBooks = async (req, res) => {
9
10     const bookName = req.query.name;
11     const publishedAfter = req.query.publishedAfter;
12     let promise;
13
14     if ( bookName && publishedAfter ) {
15         const publishedAfterDate = new Date(publishedAfter);
16
17         promise = Books.findAll({
18             where: {
19                 title: {
20                     [Op.substring]: bookName
21                 },
22                 published: {
23                     [Op.gt]: publishedAfterDate
24                 },
25                 authorId: req.user
26             }
27         });
28     }
29     else if ( publishedAfter ) {
30         const publishedAfterDate = new Date(publishedAfter);
31
32         promise = Books.findAll({
33             where: {
34                 published: {
35                     [Op.gt]: publishedAfterDate
36                 },
```

JS bookModel.js M

JS bookController.js M X

controllers > JS bookController.js > [x] getAllBooks > where

```
35         [Op.gt]: publishedAfterDate
36     },
37     authorId: req.user
38 }
39 });
40 }
41 else if ( bookName ) {
42     promise = Books.findAll({
43         where: {
44             title: {
45                 [Op.substring]: bookName
46             },
47             authorId: req.user
48         },
49
50     });
51 } else {
52     promise = Books.findAll({
53         where: {
54             authorId: req.user
55         }
56     });
57 }
58
59 promise
60     .then((books) => {
61         res.send(books);
62     })
63     .catch(() => {
64         console.error(error.message);
65         res.status(500).json({message: error.message})
66     })
67
68 }
69
```

Testing APIs:

GET /books

BooksAPI / http://localhost:8080/books

GET http://localhost:8080/books

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Headers 7 hidden

	KEY	VALUE	DESCRIF		Bulk Edit	Presets
<input checked="" type="checkbox"/>	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...				
	Key	Value	Description			

Body Cookies Headers (7) Test Results 200 OK 29 ms 421 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "title": "Open Ride",
5     "price": 599.99,
6     "author": "Kumar Mohit",
7     "authorId": 1,
8     "published": "2022-02-26",
9     "publisher": "AMD Book House",
10    "createdAt": "2022-02-26",
11    "updatedAt": "2022-02-26"
12  }
13 ]
```

GET /books/:id

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

- Request Method:** GET
- Request URL:** http://localhost:8080/books/1
- Headers:** x-auth-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...
- Response Status:** 200 OK, 33 ms, 419 B
- Response Body (JSON):**

```
{  "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.