

## **Module 8: Final Project**

**Brandon Trinkle**

**IFT 458: Middleware Programming**

**Professor Dinesh Sthapit**

**April 26, 2025**

## Module 8: Final Project

This project implements a secure, middleware-driven Node.js application for a Book Exchange platform. It leverages Express.js for routing, Mongoose for MongoDB interactions, HTTPS for transport security, and Docker for containerization. Another term for this stack is called MERN. “MERN Stack is a JavaScript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises of 4 technologies namely: MongoDB, Express, React and NodeJS. It is designed to make the development process smoother and easier.” (Geek for Geeks, 2025)

### Problem Statement

- Design and deploy a robust backend that enforces authentication, input validation, error handling, and secure communication to support a book exchange service.

### Justification of Technology Stack

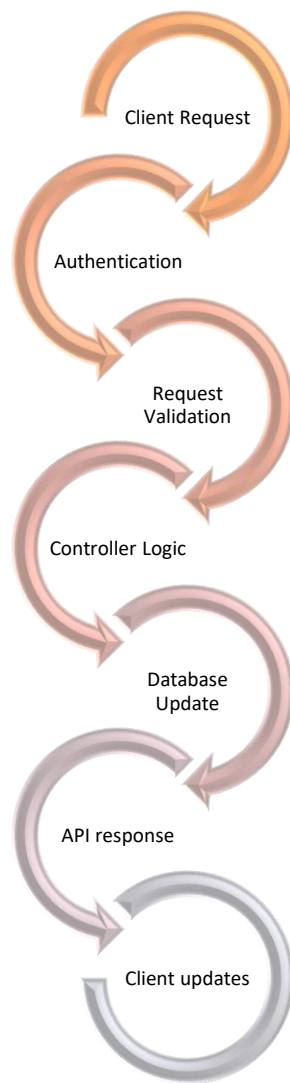
- Node.js with Express.js: provides high-performance, event-driven server capabilities.
- MongoDB with Mongoose: flexible schema design for book and user data.
- HTTPS via self-signed certificates ensures encrypted communication during development and testing.
- Docker: isolates the application environment for consistent deployments.

## Project Design

In this design, routes serve as the entry point by mapping incoming HTTP requests to controller functions. Controllers implement the application’s business logic: they process client requests, invoke model operations for data retrieval or updates, and then select the appropriate

views for response. Models encapsulate schema definitions and interact with MongoDB through Mongoose, ensuring data validation and persistence. Views consist of EJS templates that render dynamic HTML by injecting data provided by controllers. This separation concerns streamlines development and testing, as each component's responsibilities are clearly defined, improving maintainability and scalability.

Figure 1: Flow Chart



*Figure 1*

## API Endpoints:

Endpoint	Method	Description
/api/signup	POST	Register new user
/api/login	POST	Authenticate and issue token
/api/books	GET	Retrieve all books
/api/books	POST	Add a new book exchange
/api/books/:id	PUT	Update book by ID
/api/books/:id	DELETE	Delete book by ID

Figure 2 Shows custom middleware logs each request and enforces authentication.

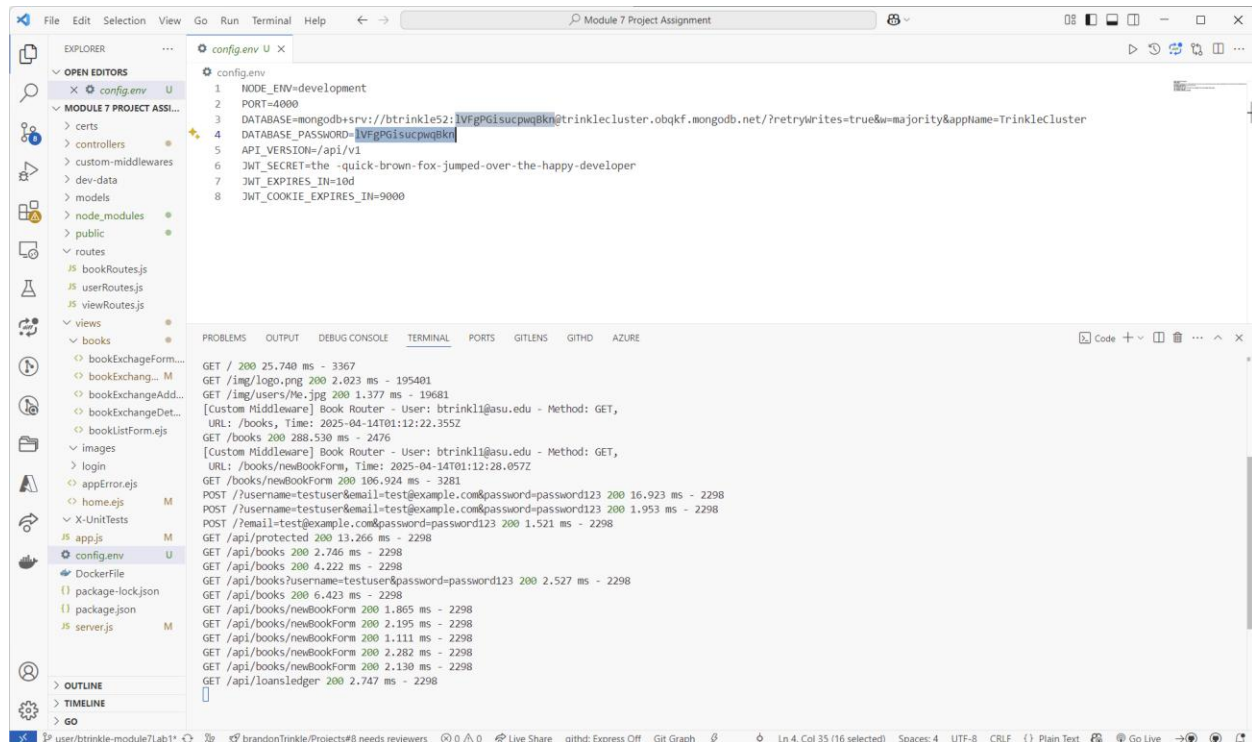


Figure 2

## **Learning Outcomes**

Through Modules 5 to 8, we worked on developing a robust, hands-on understanding of middleware design, secure communication, and containerized deployment within a Node.js/Express.js environment. In module 6 we focused on building advanced routing techniques and custom middleware chains for input validation and error handling. During the Docker exercises we built a DockerFile for our docor container. This help illustrate how to manage container lifecycles. Next, we generated SSL certificates with OpenSSL, and configuring an HTTPS server reinforced the practical nuances of transport-level security. Implementing RESTful API endpoints that returned HTTP status codes, alongside asynchronous try/catch error management, highlighted the importance of API reliability and clear client feedback. Finally, integrating authentication middleware, defining Mongoose schemas, and assembling a full MERN-stack application highlighted my learning. I can now enforce secure communication, validate user input, and deploy a containerized backend that directly aligns to industry best practices. Beyond technical competencies, these modules built systematic debugging strategies, strengthened my capacity to document workflows clearly, and instilled a security-first mindset that will guide my future software development endeavors.

## Screenshots from Labs

The screenshot shows the VS Code editor with the file `bookRoutes.js` open. The file contains the following code:

```

4 const authenticateAuthorize = require('../custom-middleware/authMiddleware');
5
6 // Authenticate middleware
7 const authenticate = authenticateAuthorize.authenticate;
8
9 // Authorize middleware;
10 const authorize = authenticateAuthorize.authorize;
11
12 // Render a new book Form and send the form to the client (browser) so that the user can send the data
13 router.get('/newBookForm', authenticate, authorize, booksController.addBook);
14
15 // CREATE is new book via a POST
16 // note that the authenticate and authorize middleware are passed as arguments
17 // the node framework allows you to pass as many middleware as you want
18 router.post('/', authenticate, authorize, booksController.createBook);
19
20 // READ is done via GET
21 // note that the authenticate and authorize middleware are passed as arguments
22 // the node framework allows you to pass as many middleware as you want
23 router.get('/', authenticate, authorize, booksController.getBooks);
24
25 // READ: GET a specific book by ID
26 // note that the authenticate and authorize middleware are passed as arguments
27 // the node framework allows you to pass as many middleware as you want
28 router.get('/:id', authenticate, authorize, booksController.getBookById);
29
30 // UPDATE is done via PUT (update) an existing book by ID
31 // Update book Form
32 router.get('/edit/:id', authenticate, authorize, booksController.updateBookForm);
33 // UPDATE i update
34 // note that the authenticate and authorize middleware are passed as arguments

```

The terminal output shows the following logs:

```

DB connection successful
[Custom Middleware] Home Route - User: Guest - Method: GET, URL: /, Time: 2025-04-14T01:12:19.181Z
GET / 200 25.740 ms - 3367
GET /img/logo.png 200 2.023 ms - 195401
GET /img/users/me.jpg 200 1.377 ms - 19681
[Custom Middleware] Book Router - User: btrinkle1@asu.edu - Method: GET, URL: /books, Time: 2025-04-14T01:12:22.355Z
GET /books 200 288.530 ms - 2476
[Custom Middleware] Book Router - User: btrinkle1@asu.edu - Method: GET, URL: /books/newBookForm, Time: 2025-04-14T01:12:28.057Z
GET /books/newBookForm 200 106.924 ms - 3281

```

Figure 3: Custom Middleware Response

The screenshot shows the VS Code editor with the file `app.js` open. The file contains the following code:

```

61 // Dynamic Route Validation Middleware
62 const isValidUserId = (userId) => /^[0-9a-f-A-F]{24}$/.test(userId);
63
64 Complexity is 3 Everything is cool!
65 app.param('userId', (req, res, next, userId) => {
66   console.log('[Middleware] UserId Param Validator triggered for userId: ${userId}');
67   if (!isValidUserId(userId)) {
68     console.log('[Middleware] Invalid User ID format detected.');
```

```

69     return res.status(400).send('Invalid User ID');
70   }
71   console.log('[Middleware] User ID format validated.');
```

```

72   next();
73 });
74
75 // Complex Middleware Chain (Authentication & Authorization)
76 app.get('/secure-data/:userId',
77   middlewareLogger('Auth Middleware (Authenticate)'), authMiddleware.authenticate,
78   middlewareLogger('Auth Middleware (Authorize)'), authMiddleware.authorize,
79   middlewareLogger('Data Processing Middleware'), (req, res, next) => {
80     req.processedAt = new Date();
81     next();
82   },
83   middlewareLogger('Response Middleware'), (req, res) => {
84     res.json({
85       userId: req.params.userId,
86       timestamp: req.processedAt,
87       message: 'Secure data accessed successfully.'
88     });
89   });
90

```

The terminal output shows the following logs:

```

btrinkle@DESKTOP-3BTQOE MINGW64 ~/OneDrive/Desktop/ASU_IFT/Projects/IFT_458_Middleware/Projects/TrinkleMyMainProject (user:btrinkle/Module6)
$

```

Figure 4: Of the middleware code for request validation in app.js.

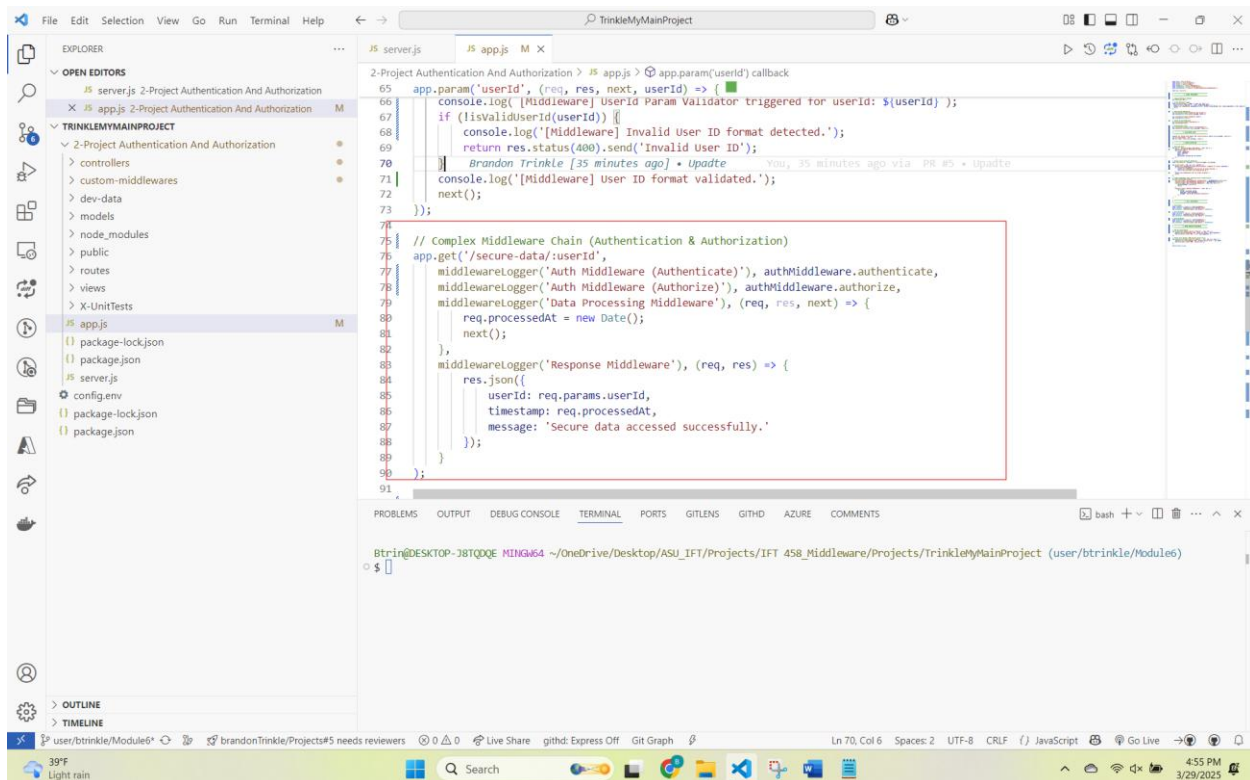


Figure 5: Showing a route in `app.js` with a complex middleware chain.

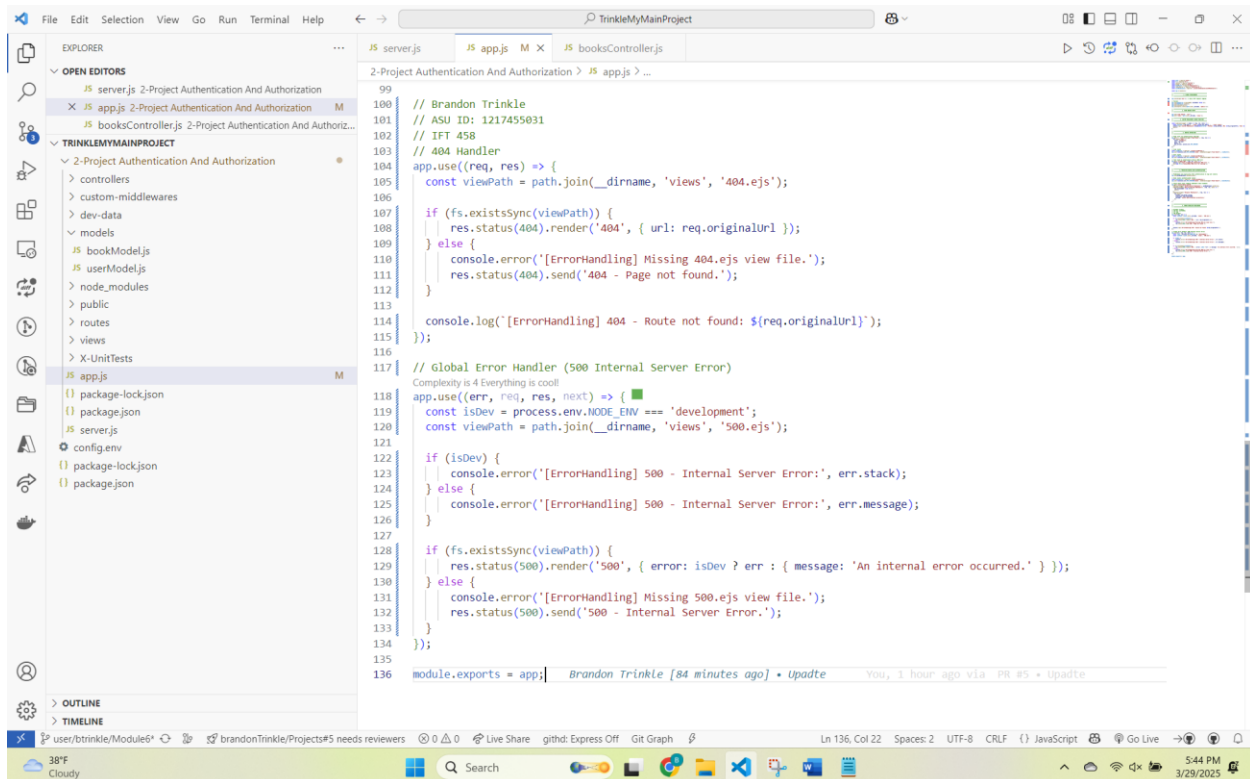


Figure 6: Of the error handling middleware in `app.js`.

Note: During testing it outputs a lot of nuisance logs – I did my best to limit the number of logs.

The screenshot displays the Visual Studio Code interface for a project named 'TrinkleMyMainProject'. The Explorer pane on the left shows the project structure, including files like 'server.js', 'app.js', and various models. The main editor shows the 'server.js' file with the following code:

```

1 // server.js
2
3 // Set up global punycode so that upstream dependencies use the userland version
4 global.punycode = require('punycode'); 217 (gzipped: 170)
5
6 // Load environment variables early
7 const dotenv = require('dotenv'); 6.3k (gzipped: 2.8k)
8 dotenv.config({ path: './config.env' });
9
10 // Import mongoose and configure strictQuery to suppress the warning
11 const mongoose = require('mongoose'); 832.8k (gzipped: 223.7k)
12 mongoose.set('strictQuery', false);

```

The Terminal pane at the bottom shows the command prompt output, including the command to run the app and the resulting logs showing successful database connection and app startup on port 4000.

```

$ node "c:\Users\Btr\OneDrive\Desktop\ASU_IPT\Projects\IFT 458_Middleware\Projects\TrinkleMyMainProject\2-Project Authentication And Authorization\server.js"
[Setup] View Engine: EJS, Views Directory: c:\Users\Btr\OneDrive\Desktop\ASU_IPT\Projects\IFT 458_Middleware\Projects\TrinkleMyMainProject\2-Project Authentication And Authorization\views
Connecting to DB: mongodb+srv://btrinkle52:1VFgGisucpwqBkn@trinklecluster.obqkf.mongodb.net/?retryWrites=true&majority=TrinkleCluster
(node:6088) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
DB connection successful!
App running on port 4000...
To test the IFT 458 REST App, visit: http://localhost:4000...
[Custom Middleware] BodyParser URL Encoded - Method: GET, URL: /, Time: 2025-03-29T21:04:51.309Z
[Custom Middleware] BodyParser JSON - Method: GET, URL: /, Time: 2025-03-29T21:04:51.310Z
[Custom Middleware] CookieParser - Method: GET, URL: /, Time: 2025-03-29T21:04:51.311Z
[Custom Middleware] Static Files Middleware - Method: GET, URL: /, Time: 2025-03-29T21:04:51.311Z
[Custom Middleware] Home Route - Method: GET, URL: /, Time: 2025-03-29T21:04:51.314Z
GET / 304 15.137 ms - -

```



Figure 7: Logging in (Auth Middleware)

The screenshot shows the VS Code editor with the `server.js` file open. The file contains the following code:

```

1 // server.js
2
3 // Set up global punycode so that upstream dependencies use the userland version
4 global.punycode = require('punycode'); // 217 (gzipped: 170)
5
6 // Load environment variables early
7 const dotenv = require('dotenv'); // 6.3k (gzipped: 2.8k)
8 dotenv.config({ path: './config.env' });
9
10 // Import mongoose and configure strictQuery to suppress the warning
11 const mongoose = require('mongoose'); // 832.8k (gzipped: 223.7k)
12 mongoose.set('strictQuery', false);
13
14 // Import your Express app
15 const app = require('./app'); // Brandon Trinkle [2 weeks ago] • Update You, 2 weeks ago • update
16
17 // Define your MongoDB connection string (or use one from your environment variables)
18 const DB =
19   process.env.DATABASE ||
20   "mongodb+srv://btrinkle52:lvfGpGisucpwqBkn@trinklecluster.obqkf.mongodb.net/?retryWrites=true&w=majority&appName=Trink
21
22 console.log("Connecting to DB:", DB);

```

The terminal output shows the following logs:

```

DB connection successful!
App running on port 4000...
To test the IFT 458 REST App, visit: http://localhost:4000...
[Custom Middleware] Home Route - User: Guest - Method: GET, URL: /, Time: 2025-03-29T21:15:34.461Z
GET / 304 28.516 ms - -
[Custom Middleware] View Router - User: Guest - Method: GET, URL: /api/v1/views/loginUser, Time: 2025-03-29T21:15:38.333Z
GET /api/v1/views/loginUser 304 2.791 ms - -
[Custom Middleware] User Router - User: Guest - Method: POST, URL: /api/v1/users/login, Time: 2025-03-29T21:15:47.773Z
POST /api/v1/users/login 200 325.737 ms - 2691
[Custom Middleware] View Router - User: Guest - Method: GET, URL: /api/v1/views, Time: 2025-03-29T21:15:49.168Z
GET /api/v1/views 304 2.700 ms - -
[AuthMiddleware] User authenticated: btrinkle1@asu.edu
[Custom Middleware] Book Router - User: btrinkle1@asu.edu - Method: GET, URL: /api/v1/books, Time: 2025-03-29T21:16:03.390Z
[AuthMiddleware] User authenticated: btrinkle1@asu.edu
[AuthMiddleware] Authorizing user: btrinkle1@asu.edu, role: client
No books found, rendering book exchange form.
GET /api/v1/books 304 145.285 ms - -

```

Figure 8: Error handling Middleware 404

The screenshot shows the VS Code editor with the `server.js` file open. The file contains the following code:

```

22 console.log("Connecting to DB:", DB);
23
24 // Connect to the database, then start the server if successful
25 mongoose
26   .connect(DB, {
27     useNewUrlParser: true,
28     useUnifiedTopology: true
29   })
30   .then(() => {
31     console.log('DB connection successful!');
32     const port = process.env.PORT || 3000;
33     app.listen(port, () => {
34       console.log(`App running on port ${port}...`);
35       console.log(`To test the IFT 458 REST App, visit: http://localhost:${port}...`);
36     });
37   })
38   .catch(err => {
39     console.error('DB connection failed!', err);
40   });
41

```

The terminal output shows the following logs:

```

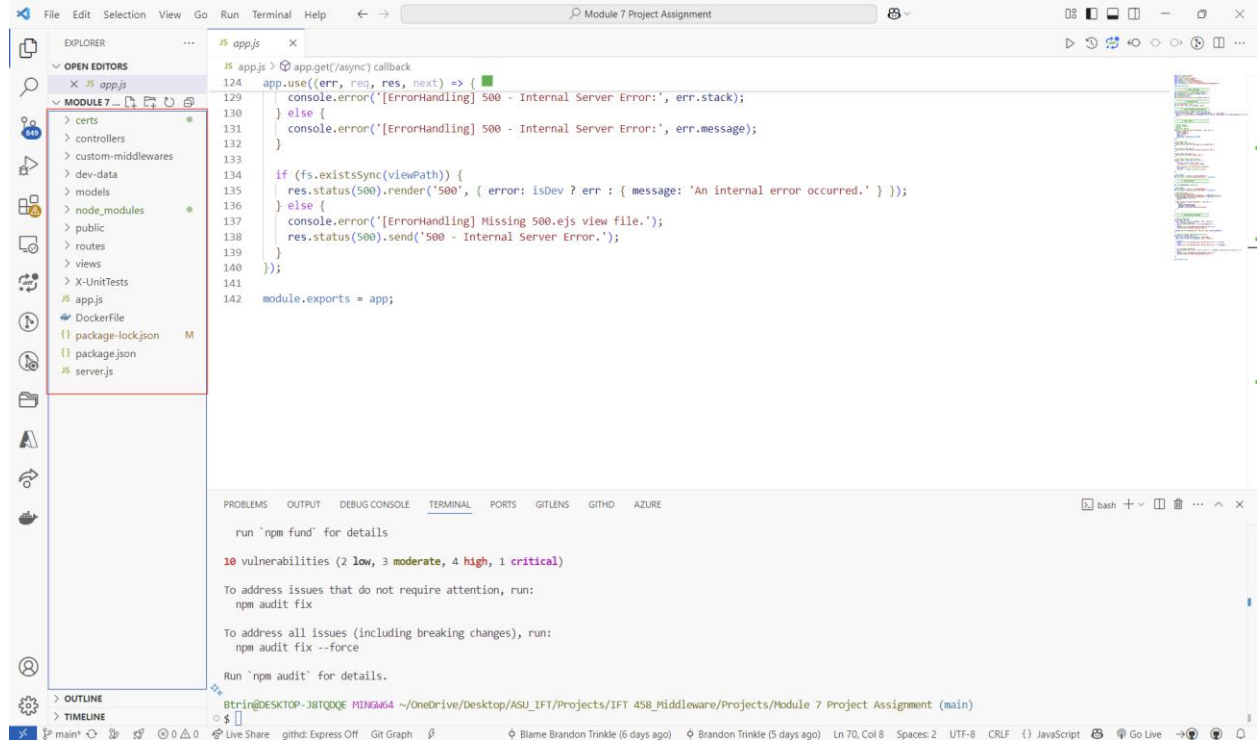
$ node "c:\Users\btrink\OneDrive\Desktop\ASU_IFT\Projects\IFT 458_Middleware\Projects\TrinkleMyMainProject\2-Project Authentication And Authoriz
ation\server.js"
Connecting to DB: mongodb+srv://btrinkle52:lvfGpGisucpwqBkn@trinklecluster.obqkf.mongodb.net/?retryWrites=true&w=majority&appName=Trinkleclust
er
(node:13592) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
DB connection successful!
App running on port 4000...
To test the IFT 458 REST App, visit: http://localhost:4000...
[AuthMiddleware] User authenticated: btrinkle1@asu.edu
[ErrorHandling] Missing 404.ejs view file.
[ErrorHandling] 404 - Route not found: /non-existent-route
GET /non-existent-route 404 63.269 ms - 21

```

Figure 9: Error handling middleware

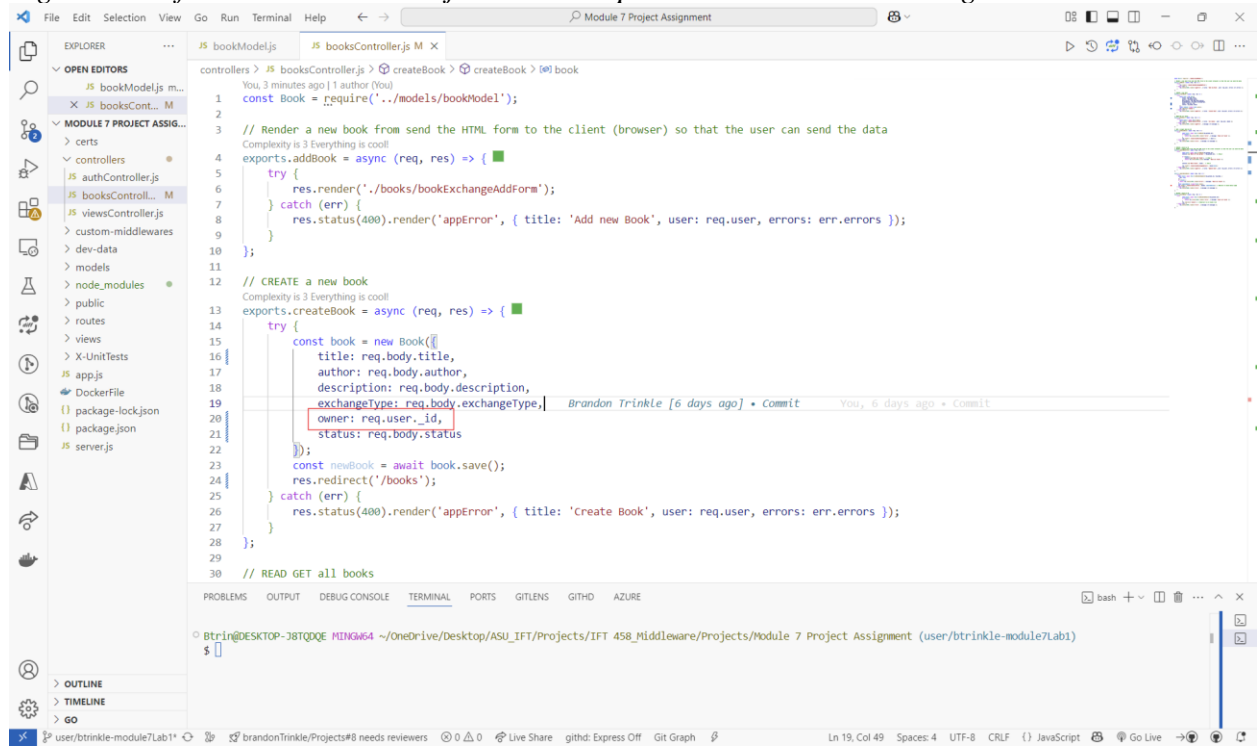
Method: Attempting to use <http://localhost:4000/non-existent-route>

Figure 10: Of the project directory structure showcasing MVC organization.



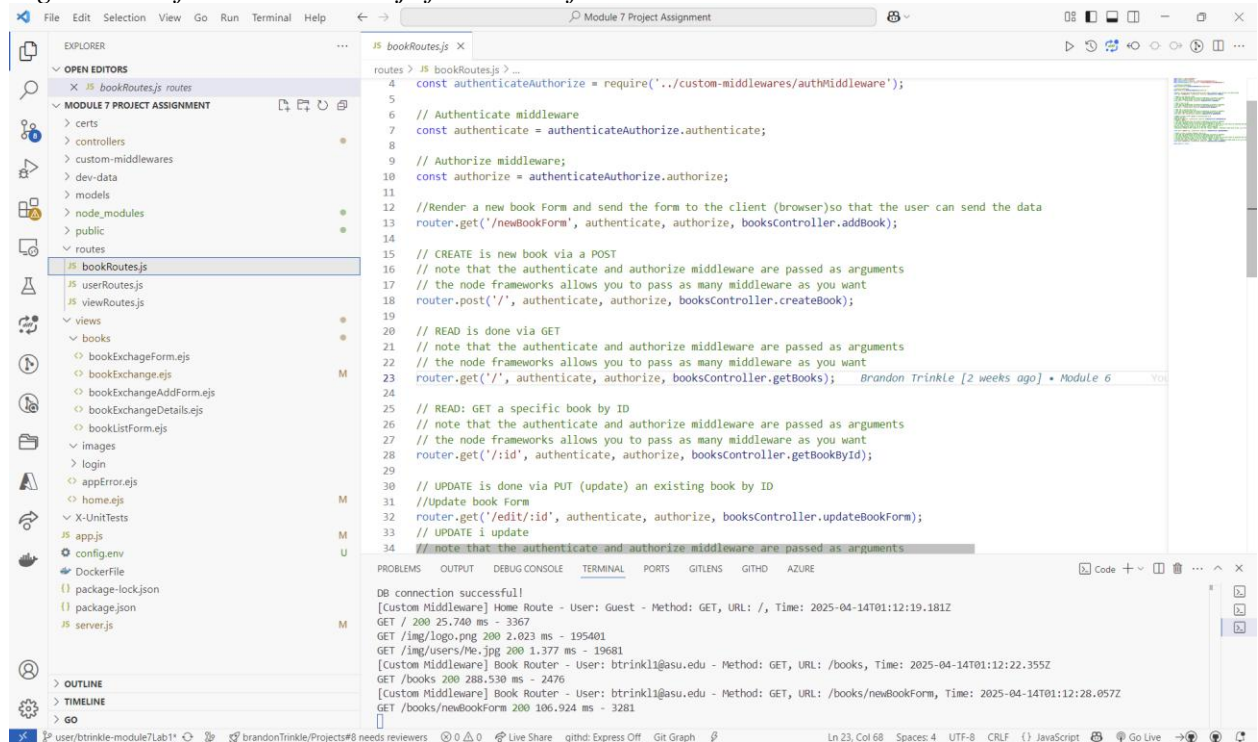
Note: This structure follows the MVC pattern. Certs created a few lessons ago are stored in the certs directory.

Figure 11: Of the bookController.js with the implemented Create Book Logic.

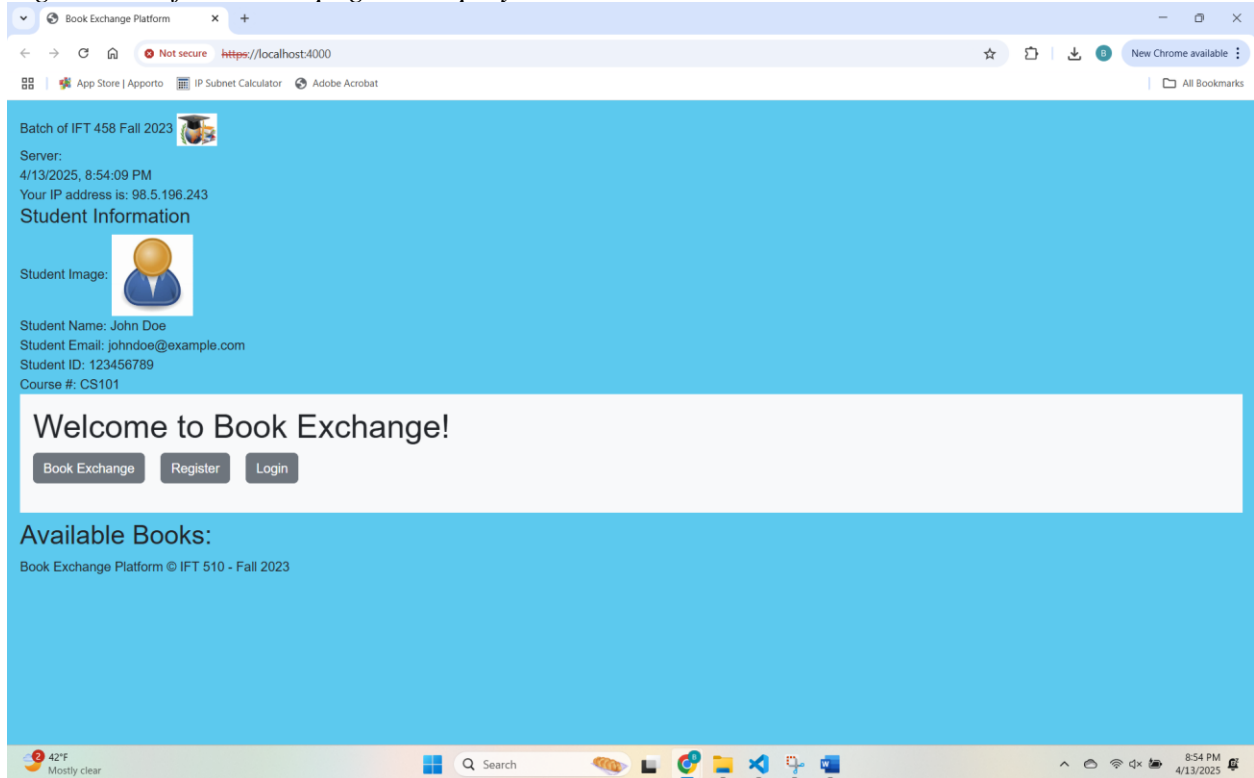


Note: owner exists already when created a new book as shown in bookController.js

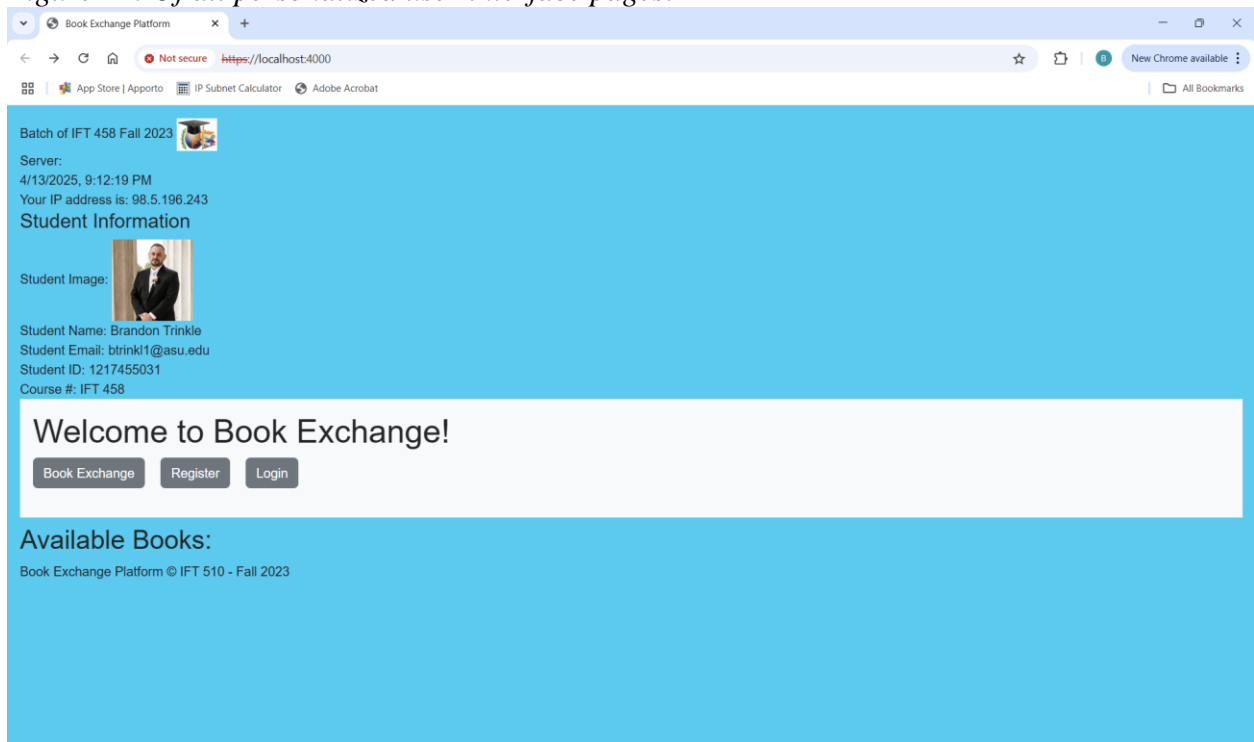
Figure 12: Of the bookRoutes.js file with defined routes.



*Figure 13: Of the home page as displayed in the browser.*



*Figure 14: Of all personalized user interface pages.*



*Note: home.ejs is the only file that is customized to the user*

Figure 15: Of the testing process and responses for different API endpoints.

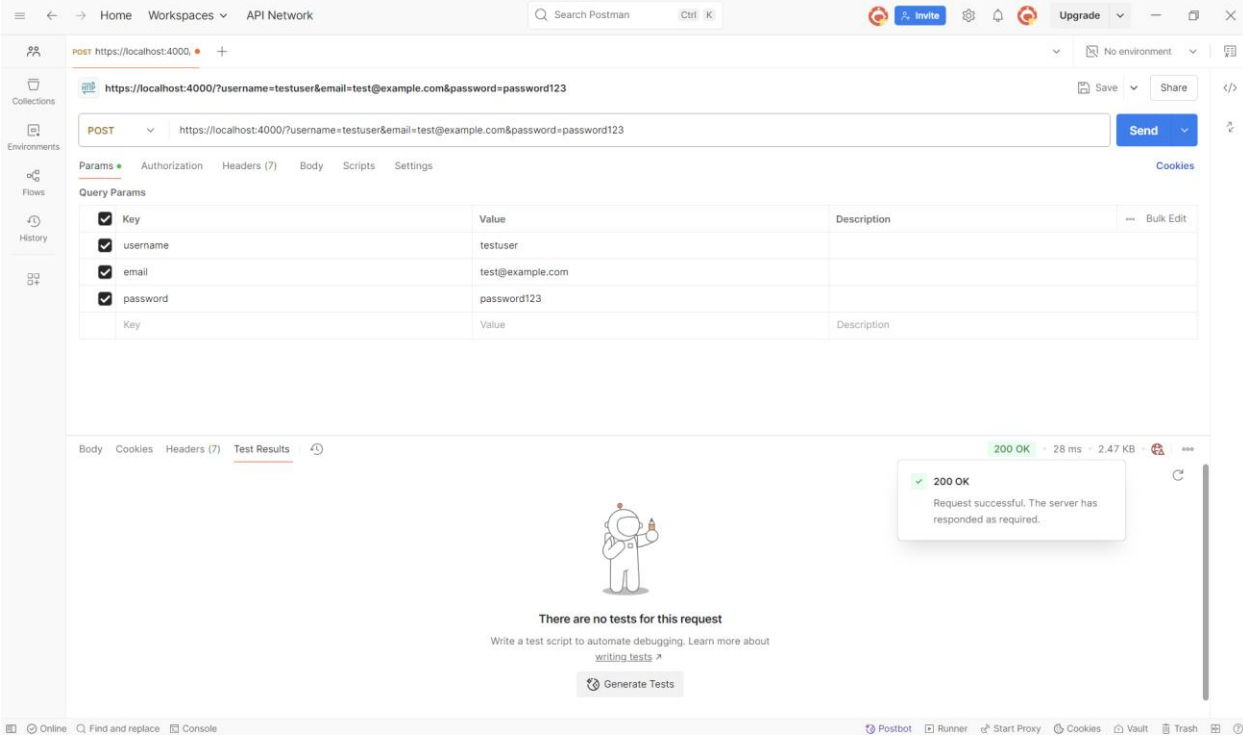


Figure 15: Of API specification documentation and screenshots of the corresponding code for each endpoint.

a. Register / Sign up

```

1  const express = require('express');
2  const router = express.Router();
3  const booksController = require('../controllers/booksController');
4  const authenticateAuthorize = require('../custom-middlewares/authMiddleware');
5
6  // Authenticate middleware
7  const authenticate = authenticateAuthorize.authenticate;
8
9  // Authorize middleware;
10 const authorize = authenticateAuthorize.authorize;
11
12 // Render a new book form and send the form to the client (browser) so that the user can send the data
13 router.get('/newBookForm', authenticate, authorize, booksController.addBook);
14
15 // CREATE is new book via a POST
16 // note that the authenticate and authorize middleware are passed as arguments
17 // the node frameworks allows you to pass as many middleware as you want
18 router.post('/', authenticate, authorize, booksController.createBook);
19
20 // READ is done via GET
21 // note that the authenticate and authorize middleware are passed as arguments
22 // the node frameworks allows you to pass as many middleware as you want
23 router.get('/newBookForm', authenticate, authorize, booksController.addBook);
24
25 // READ: GET a specific book by ID
26 // note that the authenticate and authorize middleware are passed as arguments
27 // the node frameworks allows you to pass as many middleware as you want
28 router.get('/:id', authenticate, authorize, booksController.getBookById);

```

Terminal output:

```

btrink@DESKTOP-38TQDQE MINGW64 ~/OneDrive/Desktop/ASU_IIFT/Projects/IIFT_458_Middleware/Projects/Module 7 Project Assignment (user/btrinkle-module7Lab1)
$

```

## b. Login

POST https://localhost:4000/?email=test@example.com&password=password123

Query Params:

Key	Value	Description
username	testuser	
email	test@example.com	
password	password123	

Response: 200 OK · 21 ms · 2.47 KB

There are no tests for this request

Write a test script to automate debugging. Learn more about [writing tests](#)

[Generate Tests](#)

## c. Protected Resource



GET https://localhost:4000/api/protected

Headers (7)

Key	Value	Description
Bearer	IVFgPGisucpwqBkn	
Key	Value	Description

200 OK · 25 ms · 2.47 KB

There are no tests for this request

Write a test script to automate debugging. Learn more about [writing tests](#)

[Generate Tests](#)

#### d. View All Books

GET https://localhost:4000/api/books

Scripts

```

1 pm.test("Response status code is 200", function () {
2   pm.expect(pm.response.code).to.equal(200);
3 });
4
5
6 pm.test("Response time is less than 200ms", function () {
7   pm.expect(pm.response.responseTime).to.be.below(200);
8 });
9
10
11 pm.test("Response must have the required Content-Type header", function () {
12   pm.expect(pm.response.headers.get("Content-Type")).to.include("text/html");
13 });
14
15
16 pm.test("Response body is not empty", function () {
17   pm.expect(pm.response.text()).to.not.be.empty;
18 });

```

200 OK · 24 ms · 2.47 KB

Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. [Learn more](#)

Run Collection

Filter Results

- PASSED** Response status code is 200
- PASSED** Response time is less than 200ms
- PASSED** Response must have the required Content-Type header
- PASSED** Response body is not empty
- PASSED** Response body is in valid HTML format

#### e. Add Book Exchange

GET https://localhost:4000/api/books/newBookForm

Params • Authorization • Headers (8) • Body • Scripts • Settings

Query Params

Key	Value	Description
username	testuser	
password	password123	
Key	Value	Description

Body Cookies Headers (7) Test Results (5/5)

200 OK · 10 ms · 2.47 KB

Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. [Learn more](#)

Run Collection

Filter Results

- PASSED** Response status code is 200
- PASSED** Response time is less than 200ms
- PASSED** Response must have the required Content-Type header
- PASSED** Response body is not empty
- PASSED** Response body is in valid HTML format

### f. Update Delete Exchange

GET https://localhost:4000/api/loansledger

Params • Authorization • Headers (8) • Body • Scripts • Settings

Query Params

Key	Value	Description
username	testuser	
password	password123	
Key	Value	Description

Body Cookies Headers (7) Test Results (5/5)

200 OK · 12 ms · 2.47 KB

Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. [Learn more](#)

Run Collection

Filter Results

- PASSED** Response status code is 200
- PASSED** Response time is less than 200ms
- PASSED** Response must have the required Content-Type header
- PASSED** Response body is not empty
- PASSED** Response body is in valid HTML format



*Note: We can also see that our custom middleware is logging the api requests – showing that they were made successfully without error.*

**Figure 16: API route Middleware.**

The screenshot shows the VS Code interface with the Explorer on the left, the Explorer view showing the project structure, and the Terminal view at the bottom. The Explorer view shows the project structure with folders like 'certs', 'controllers', 'custom-middleware', 'dev-data', 'models', 'node\_modules', 'public', 'routes', 'views', 'books', 'images', 'login', 'appError.ejs', 'home.ejs', 'X-UnitTests', 'app.js', 'config.env', 'DockerFile', 'package-lock.json', 'package.json', and 'server.js'. The Explorer view is currently showing the 'config.env' file, which contains the following configuration:

```
1 NODE_ENV=development
2 PORT=4000
3 DATABASE=mongodb+srv://btrinkle52:IVFgPgIsucpwqBkn@trinklecluster.obqkf.mongodb.net/?retrywrites=true&majority=appName=TrinkleCluster
4 DATABASE_PASSWORD=IVFgPgIsucpwqBkn
5 API_VERSION=/api/v1
6 JWT_SECRET=the-quick-brown-fox-jumped-over-the-happy-developer
7 JWT_EXPIRES_IN=10d
8 JWT_COOKIE_EXPIRES_IN=9000
```

The Terminal view shows the output of the application, including the following log messages:

```
GET / 200 25.740 ms - 3367
GET /img/logo.png 200 2.023 ms - 195401
GET /img/users/Me.jpg 200 1.377 ms - 19681
[Custom Middleware] Book Router - User: btrinkl1@asu.edu - Method: GET,
URL: /books, Time: 2025-04-14T01:12:22.355Z
GET /books 200 288.530 ms - 2476
[Custom Middleware] Book Router - User: btrinkl1@asu.edu - Method: GET,
URL: /books/newBookForm, Time: 2025-04-14T01:12:28.057Z
GET /books/newBookForm 200 106.924 ms - 3281
POST /username=testuser&email=test@example.com&password=password123 200 16.923 ms - 2298
POST /email=test@example.com&password=password123 200 1.953 ms - 2298
GET /api/protected 200 13.266 ms - 2298
GET /api/books 200 2.746 ms - 2298
GET /api/books?username=testuser&password=password123 200 2.527 ms - 2298
GET /api/books 200 6.423 ms - 2298
GET /api/books/newBookForm 200 1.865 ms - 2298
GET /api/books/newBookForm 200 2.195 ms - 2298
GET /api/books/newBookForm 200 1.111 ms - 2298
GET /api/books/newBookForm 200 2.282 ms - 2298
GET /api/books/newBookForm 200 2.130 ms - 2298
GET /api/loansledger 200 2.747 ms - 2298
```