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

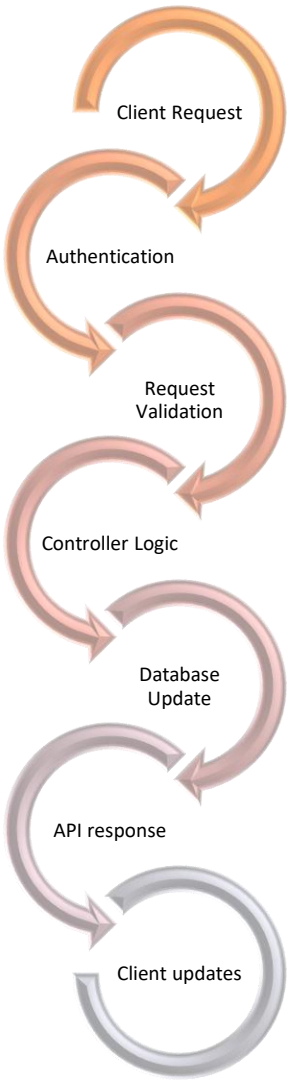


Figure 1

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.

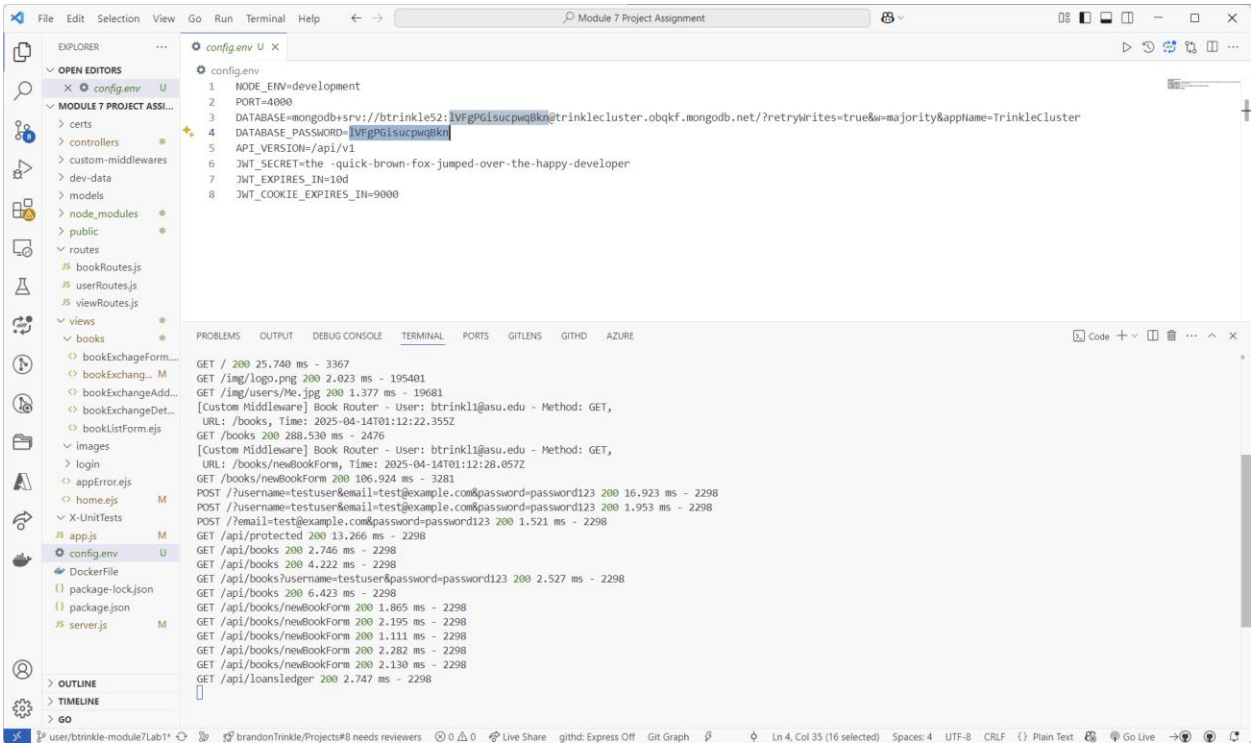


Figure 2

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

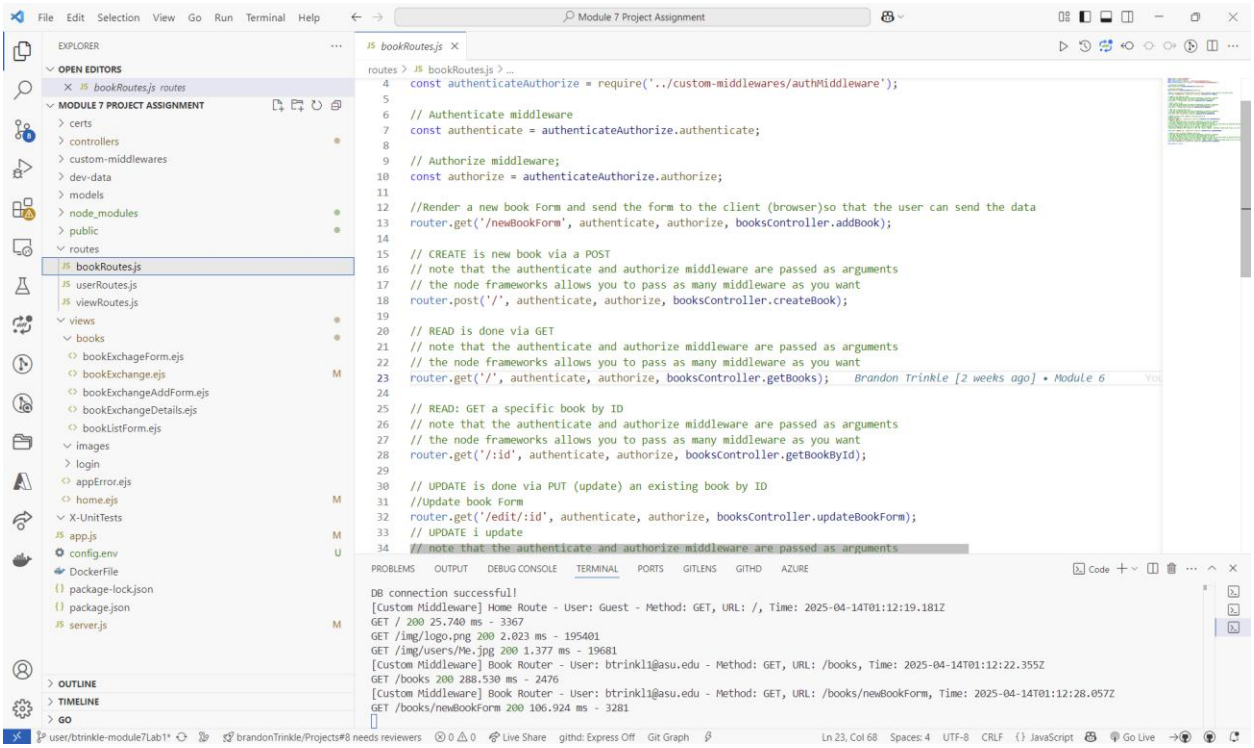


Figure 3: Custom Middleware Response

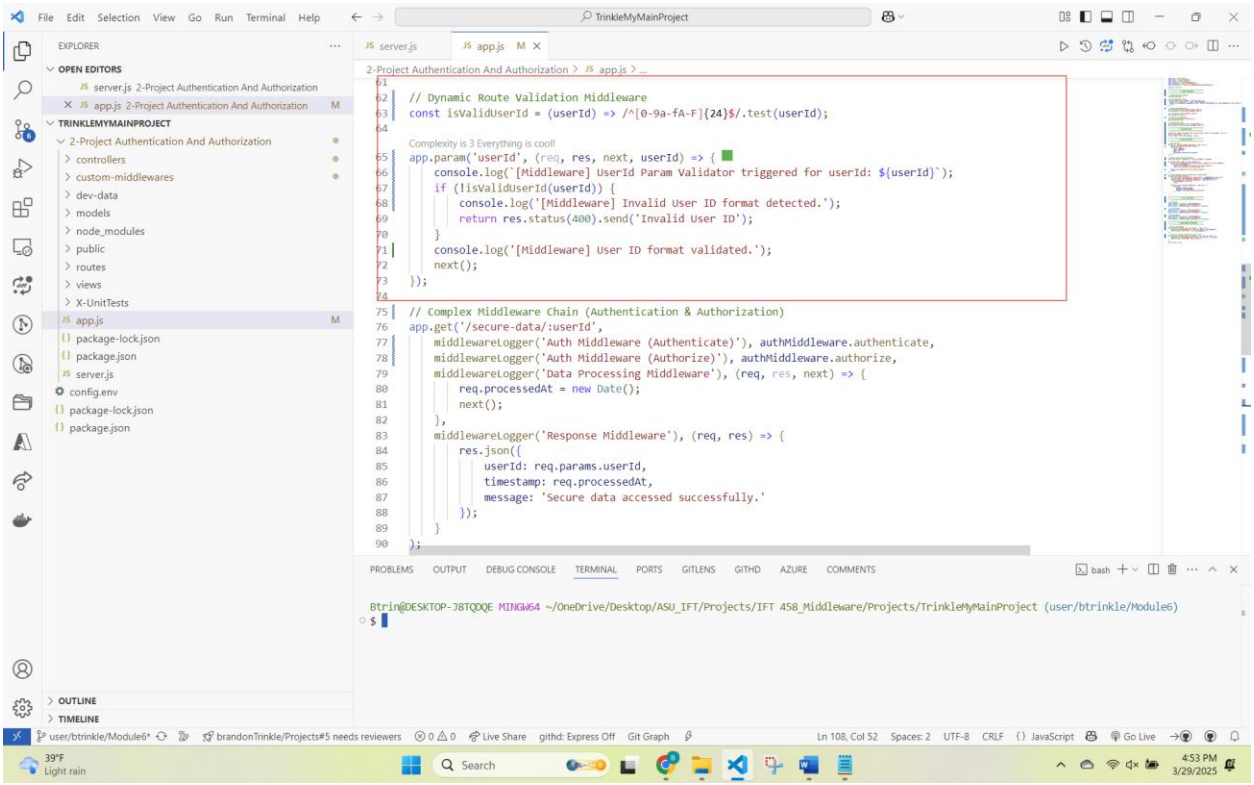


Figure 4: Of the middleware code for request validation in app.js.

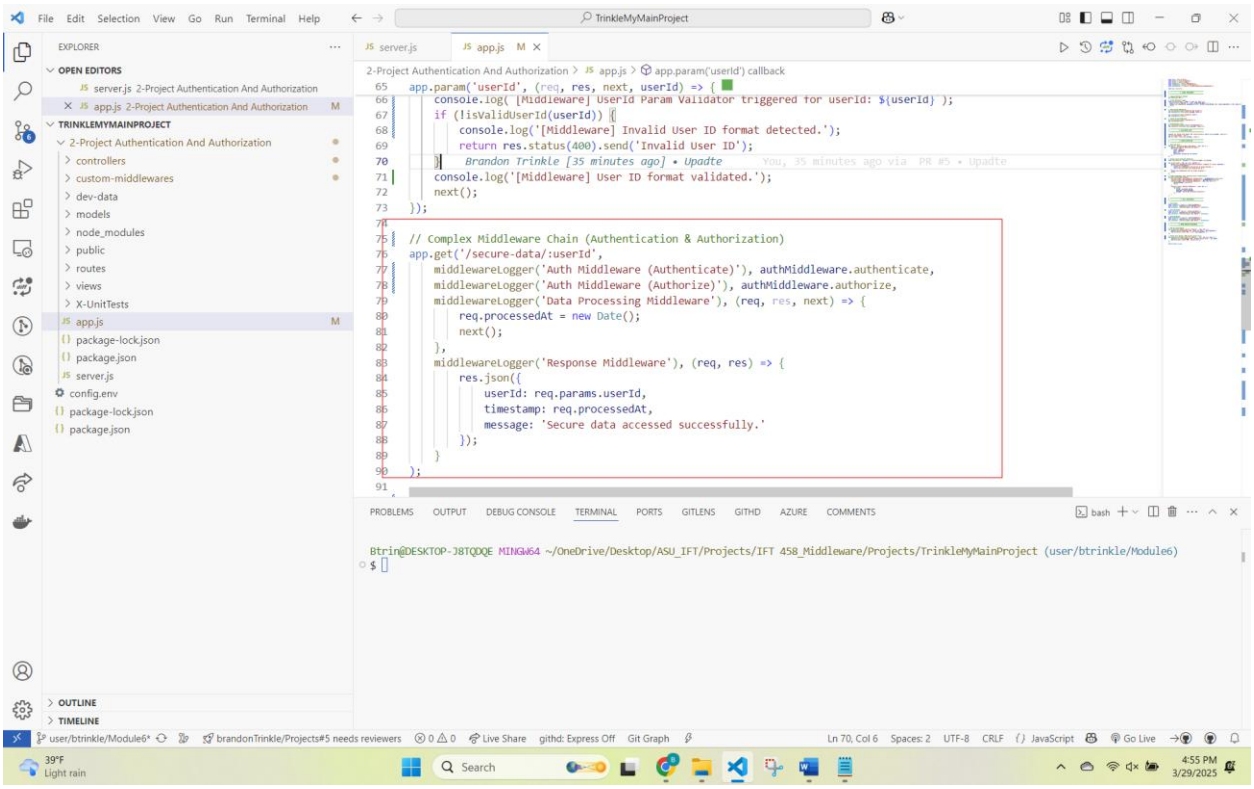


Figure 5: Showing a route in app.js with a complex middleware chain.

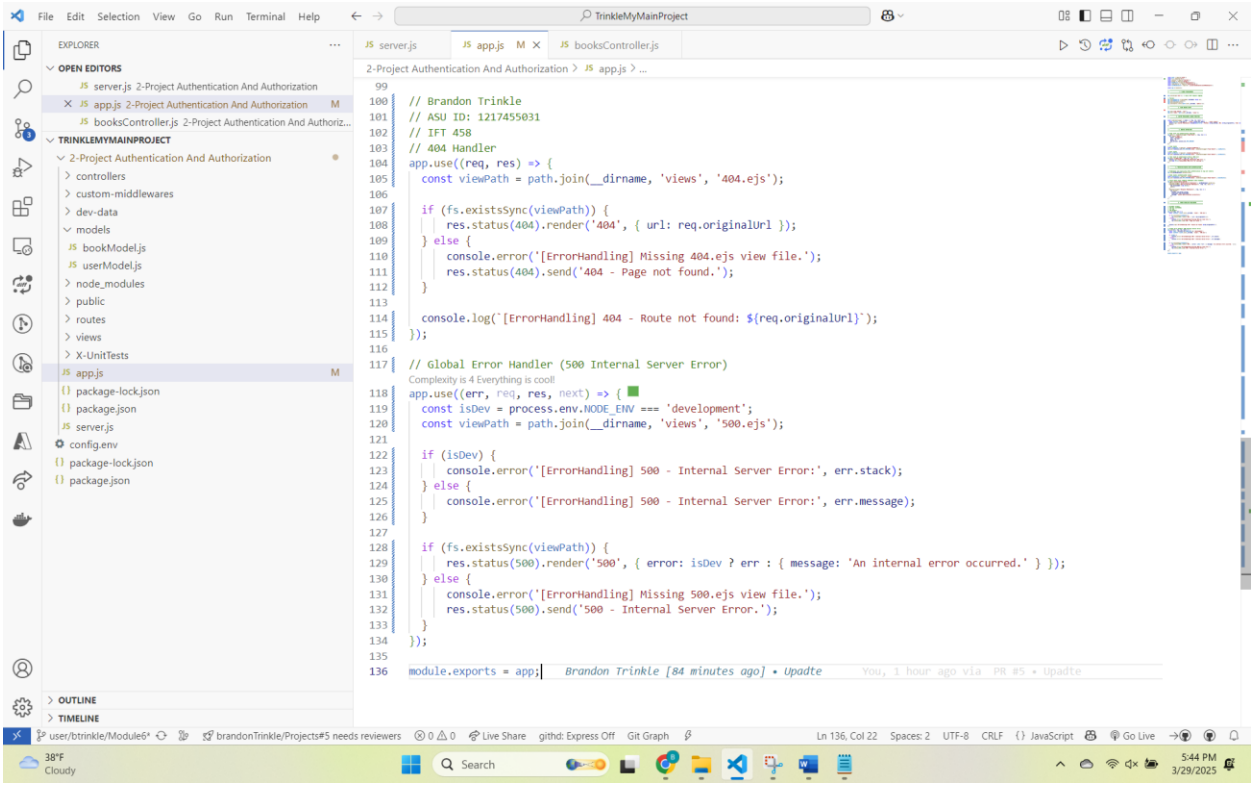


Figure 6: Of the error handling middleware in app.js.
Note: During testing it outputs a lot of nuisance logs – I did my best to limit the number of logs.

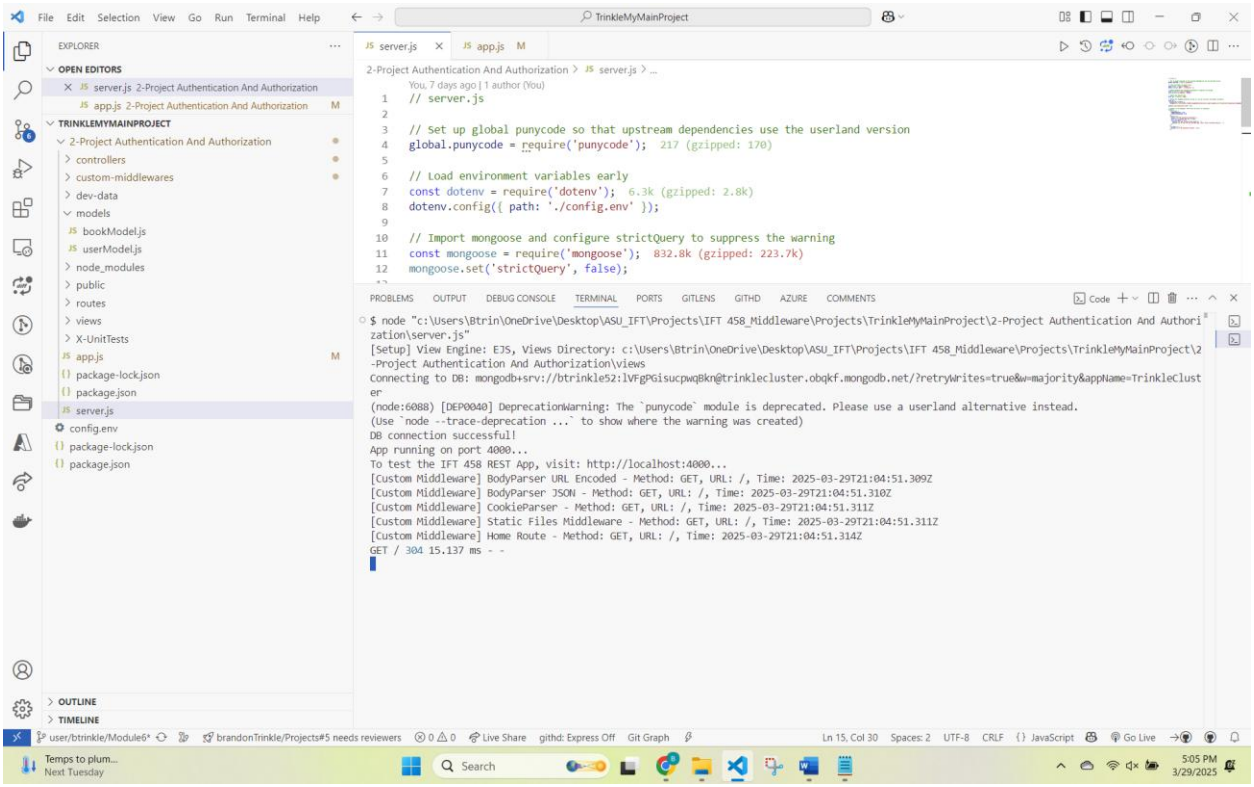

File Edit Selection View Go Run Terminal Help
TrinkleMyMainProject
EXPLORER
OPEN EDITORS
server.js 2-Project Authentication And Authorization
app.js 2-Project Authentication And Authorization M
TRINKLEMYMAINPROJECT
2-Project Authentication And Authorization
controllers
custom-middlewares
dev-data
models
bookModel.js
userModel.js
node_modules
public
routes
views
X-UnitTests
app.js M
package-lock.json
package.json
server.js
config.env
package-lock.json
package.json
server.js
app.js M
2-Project Authentication And Authorization > server.js > ...
You, 7 days ago | 1 author (You)
// server.js
// Set up global punycode so that upstream dependencies use the userland version
global.punycode = require('punycode'); 217 (gzipped: 170)
// Load environment variables early
const dotenv = require('dotenv'); 6.3k (gzipped: 2.8k)
dotenv.config({ path: './config.env' });
// Import mongoose and configure strictQuery to suppress the warning
const mongoose = require('mongoose'); 832.8k (gzipped: 223.7k)
mongoose.set('strictQuery', false);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS GITHD AZURE COMMENTS
$ node "c:\Users\Btrin\OneDrive\Desktop\ASU_IFT\Projects\IFT 458_Middleware\Projects\TrinkleMyMainProject\2-Project Authentication And Authorization\server.js"
[Setup] View Engine: EJS, Views Directory: c:\Users\Btrin\OneDrive\Desktop\ASU_IFT\Projects\IFT 458_Middleware\Projects\TrinkleMyMainProject\2-Project Authentication And Authorization\views
Connecting to DB: mongodb+srv://btrinkle52:lVFgPGisucpwqBkn@trinklecluster.obqkf.mongodb.net/?retryWrites=true&w=majority&appName=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 - -
OUTLINE
TIMELINE
user/btrinkle/Module6* brandonTrinkle/Projects#5 needs reviewers 0 0 Live Share githd: Express Off Git Graph
Ln 15, Col 30 Spaces: 2 UTF-8 CRLF {} JavaScript Go Live
Temps to plum...
Next Tuesday
Search
5:05 PM
3/29/2025

Figure 7: Logging in (Auth Middleware)

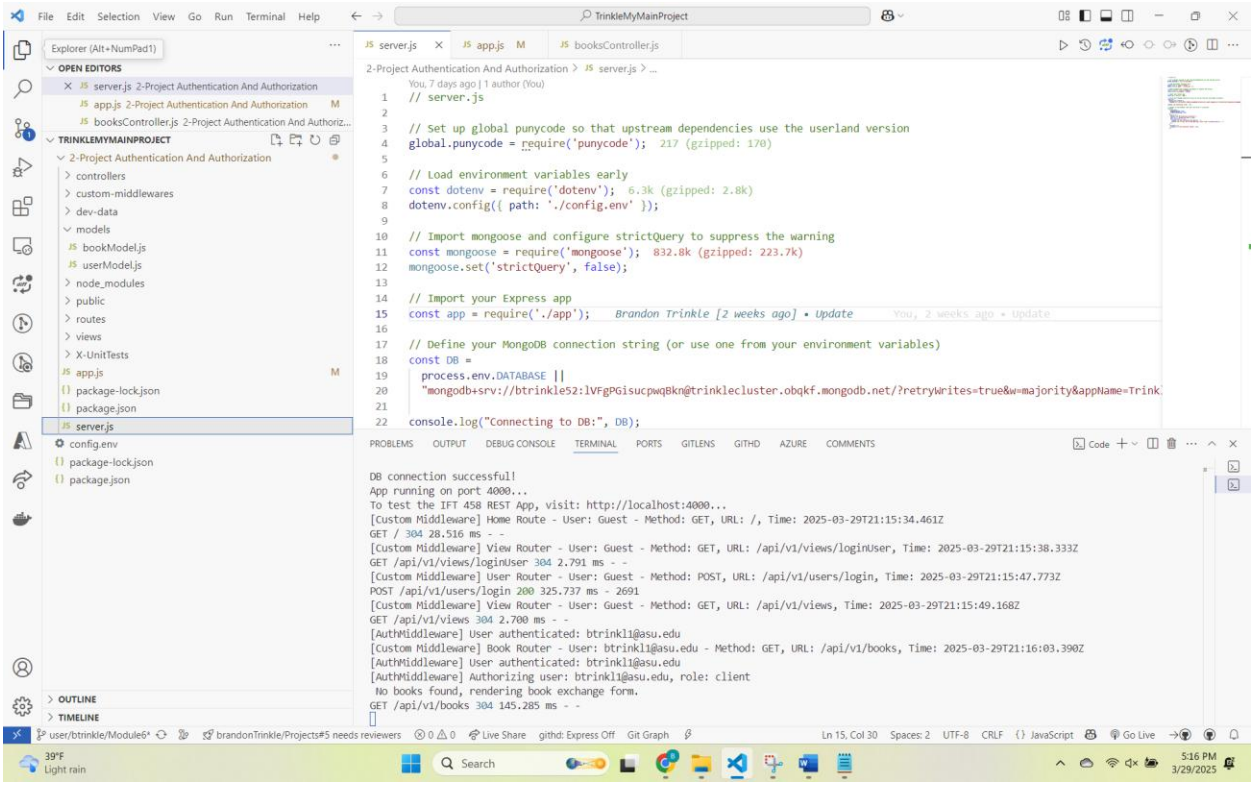


Figure 8: Error handling Middeware 404

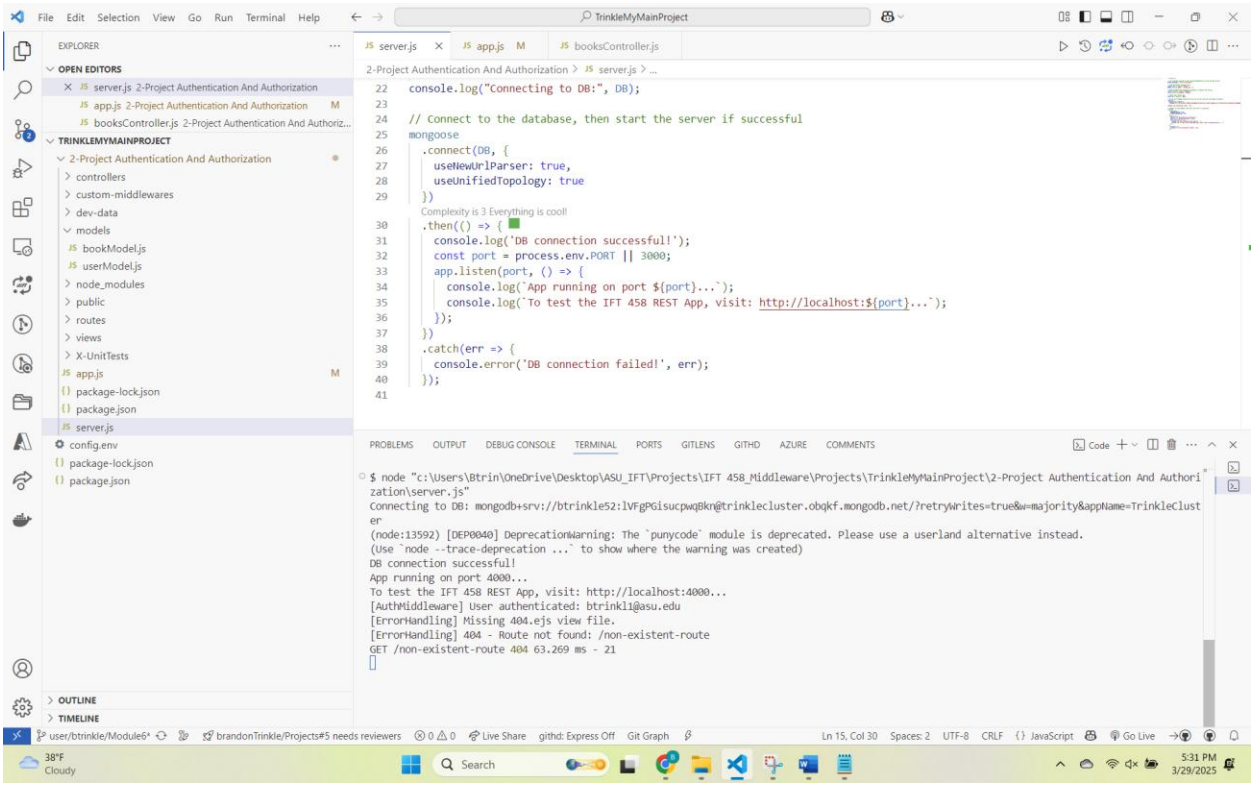
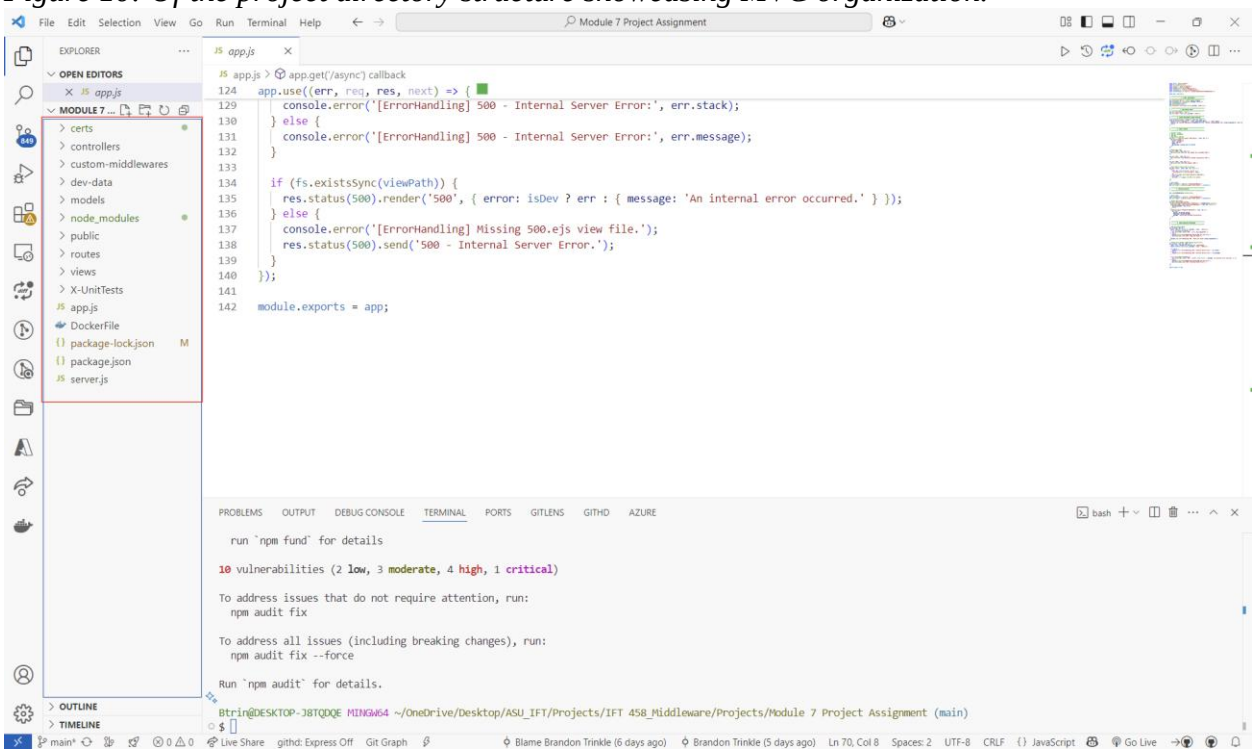


Figure 9: Error handling middleware

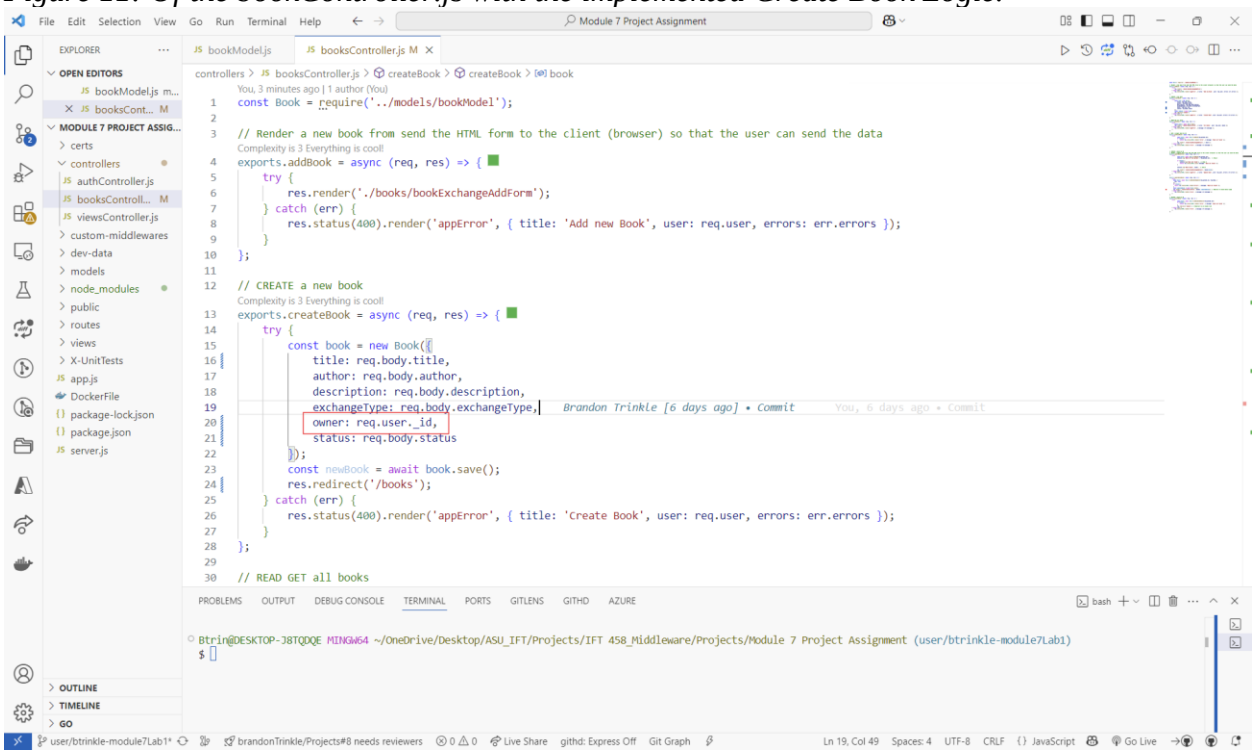
Method: Attempting to use http://localhost:4000/non-existent-route

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.

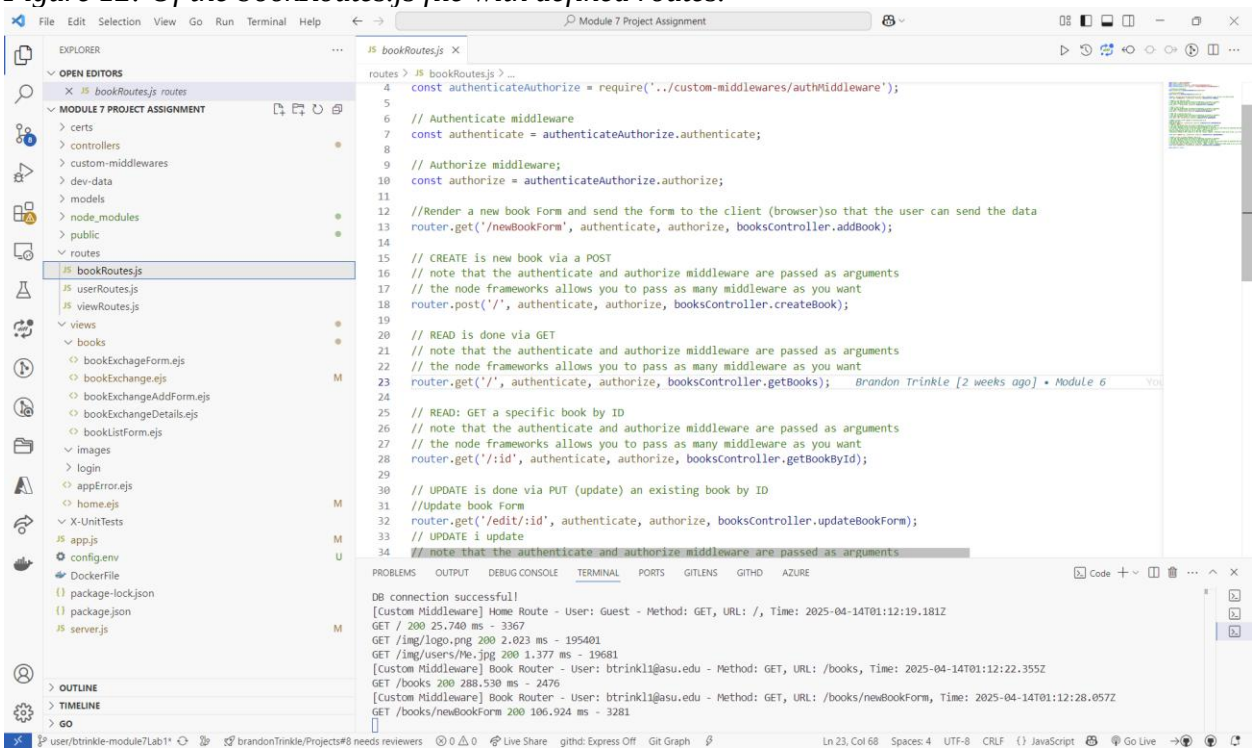


Figure 13: Of the home page as displayed in the browser.

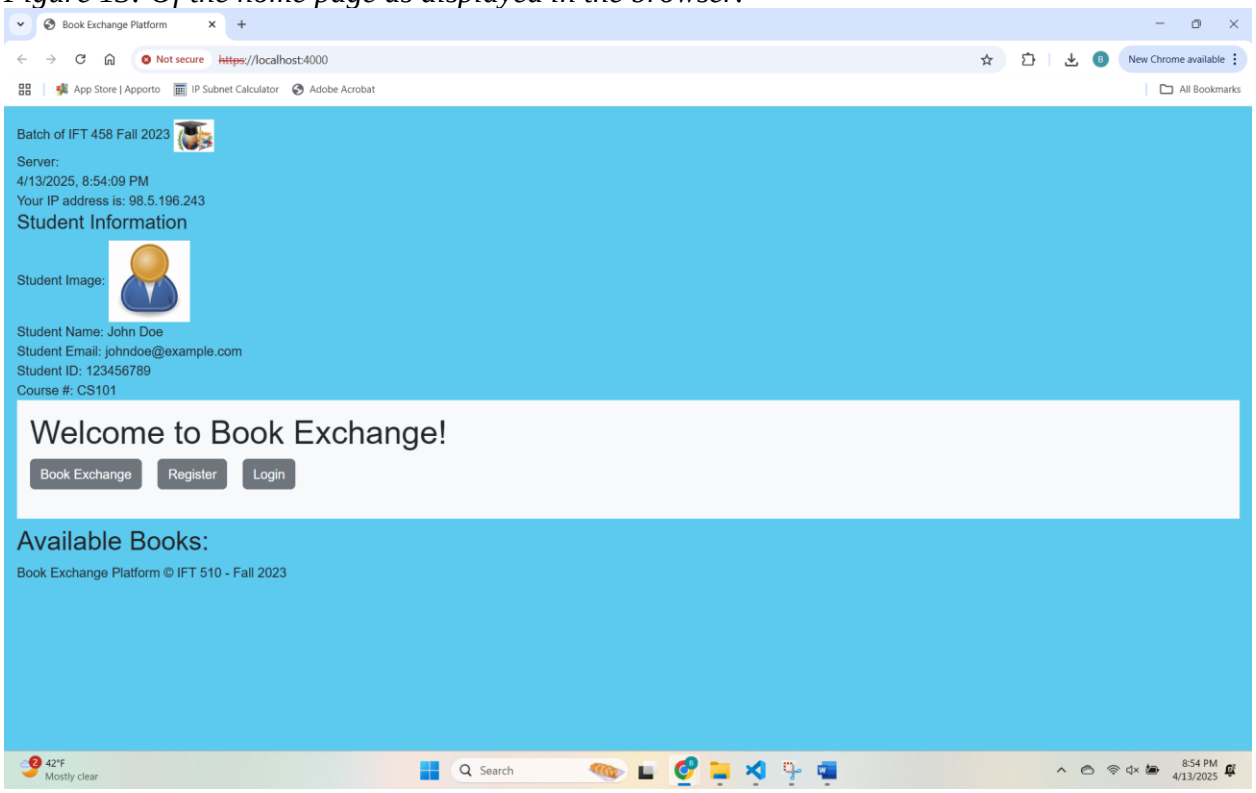
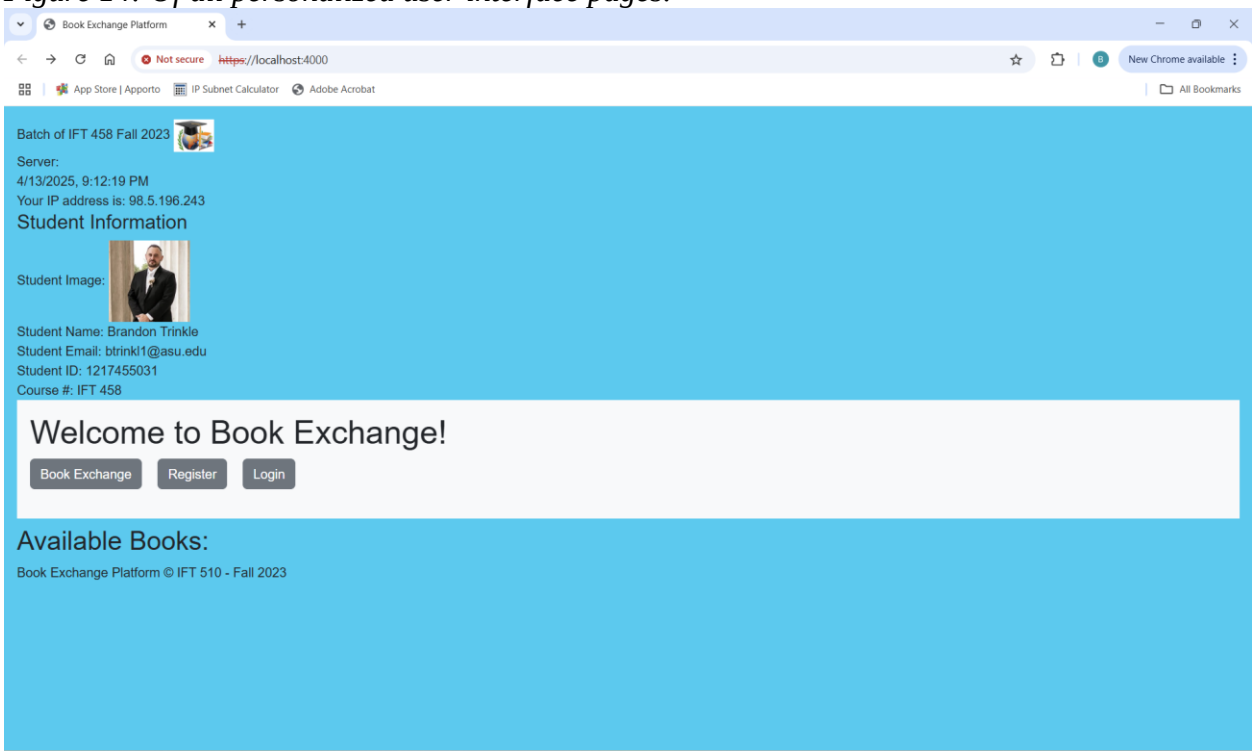


Figure 14: Of all personalized user interface pages.



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.

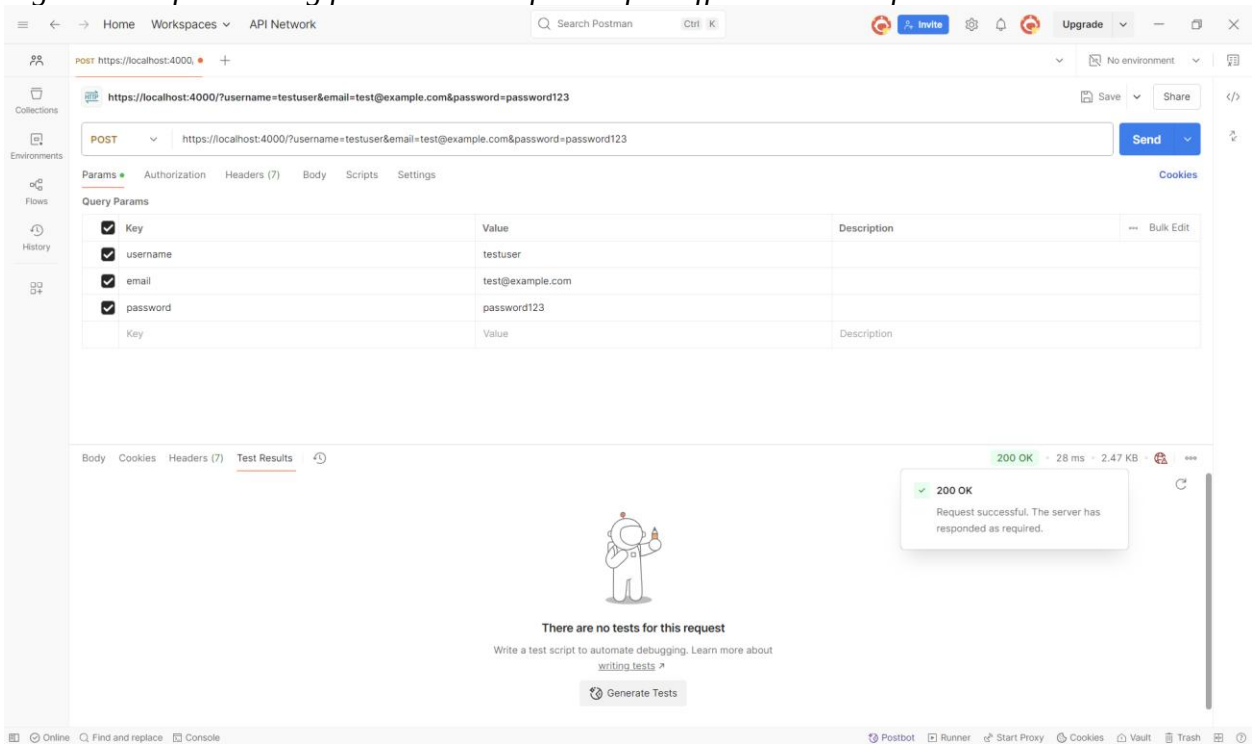
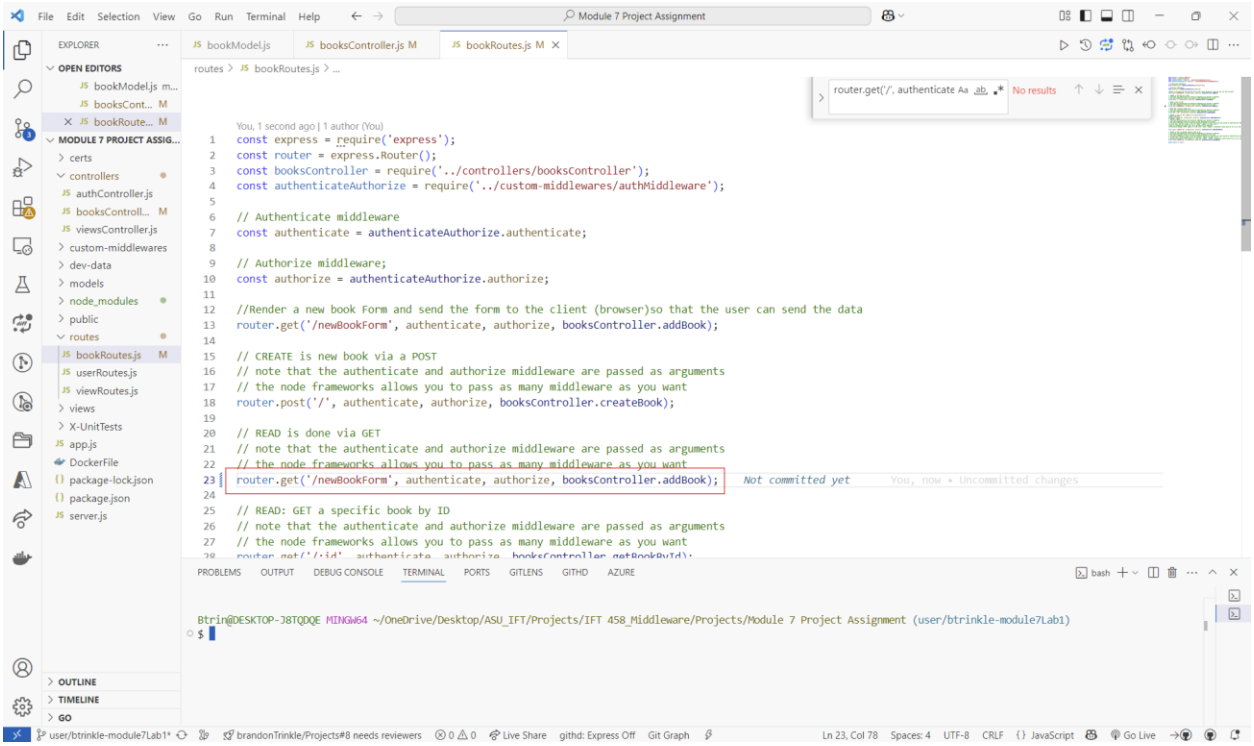
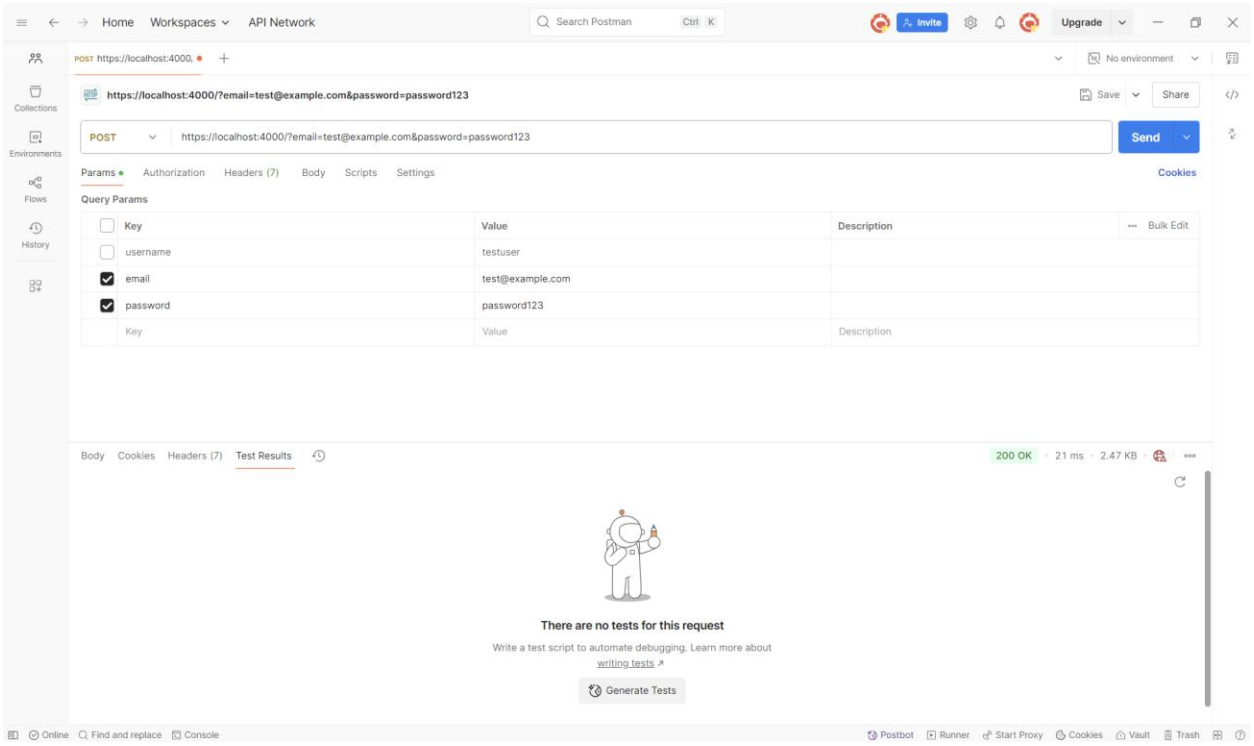


Figure 15: Of API specification documentation and screenshots of the corresponding code for each endpoint.

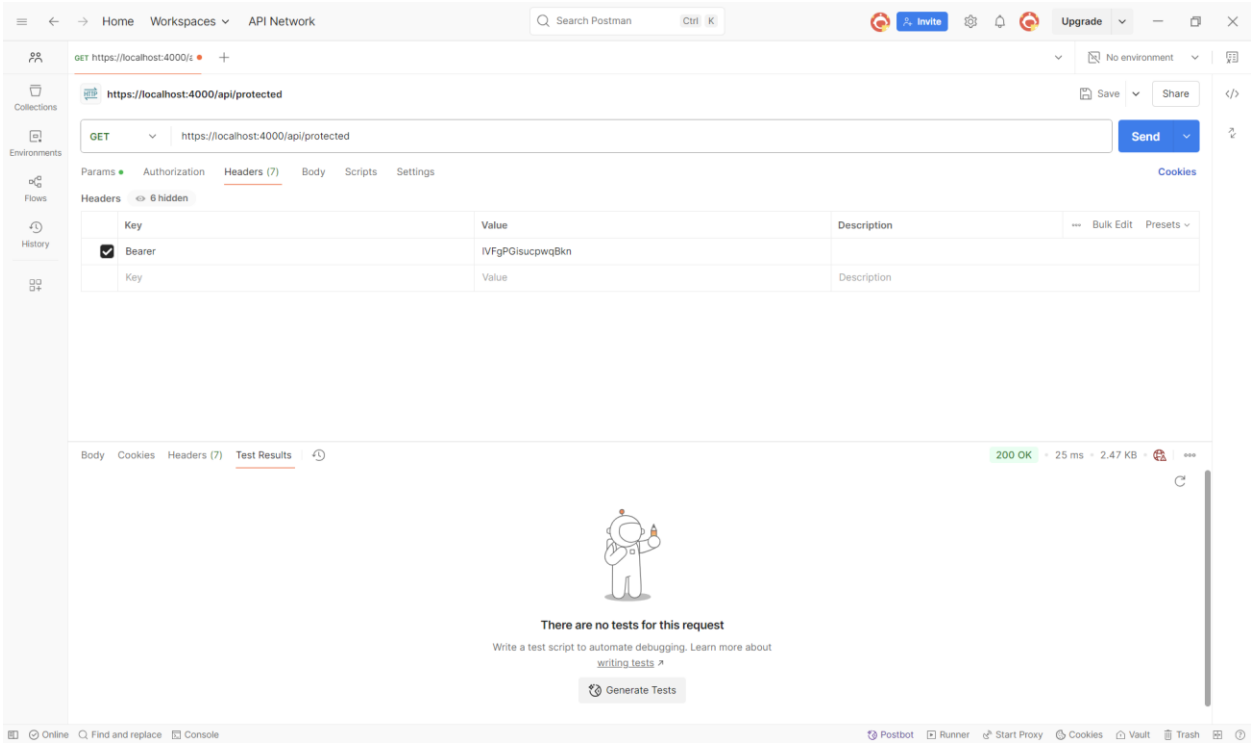
a. *Register / Sign up*



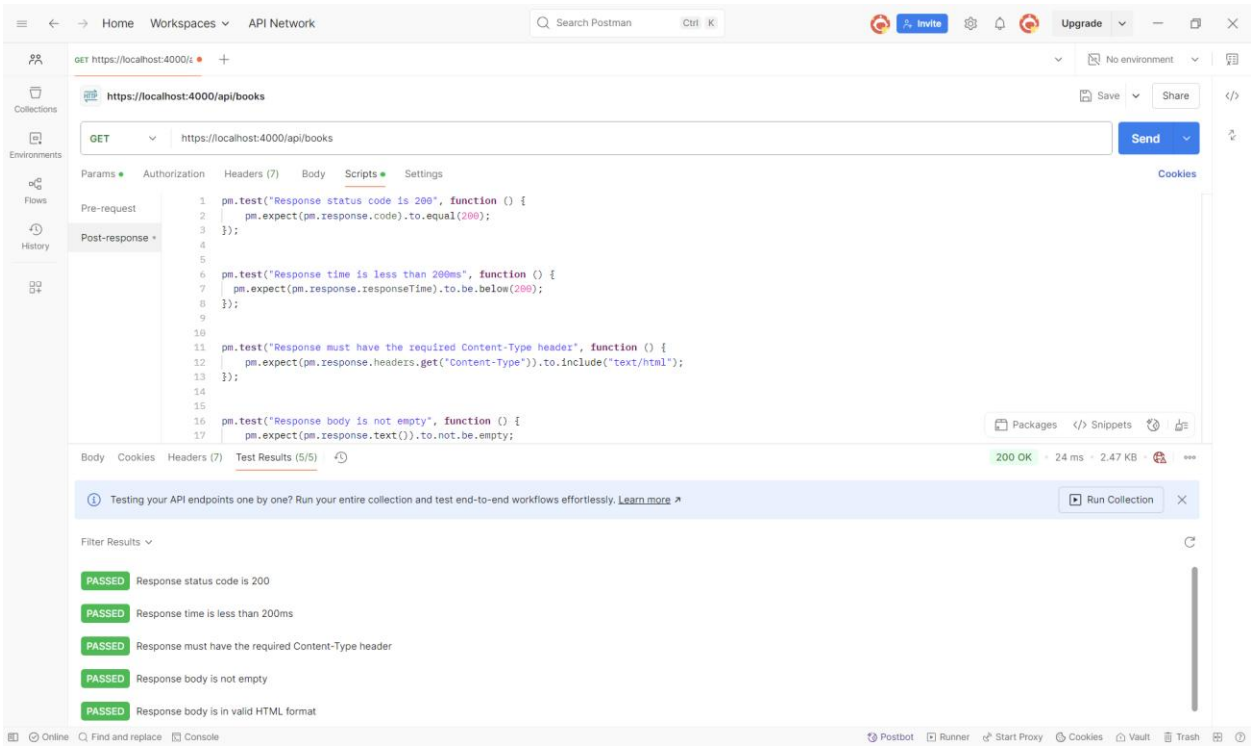
b. Login



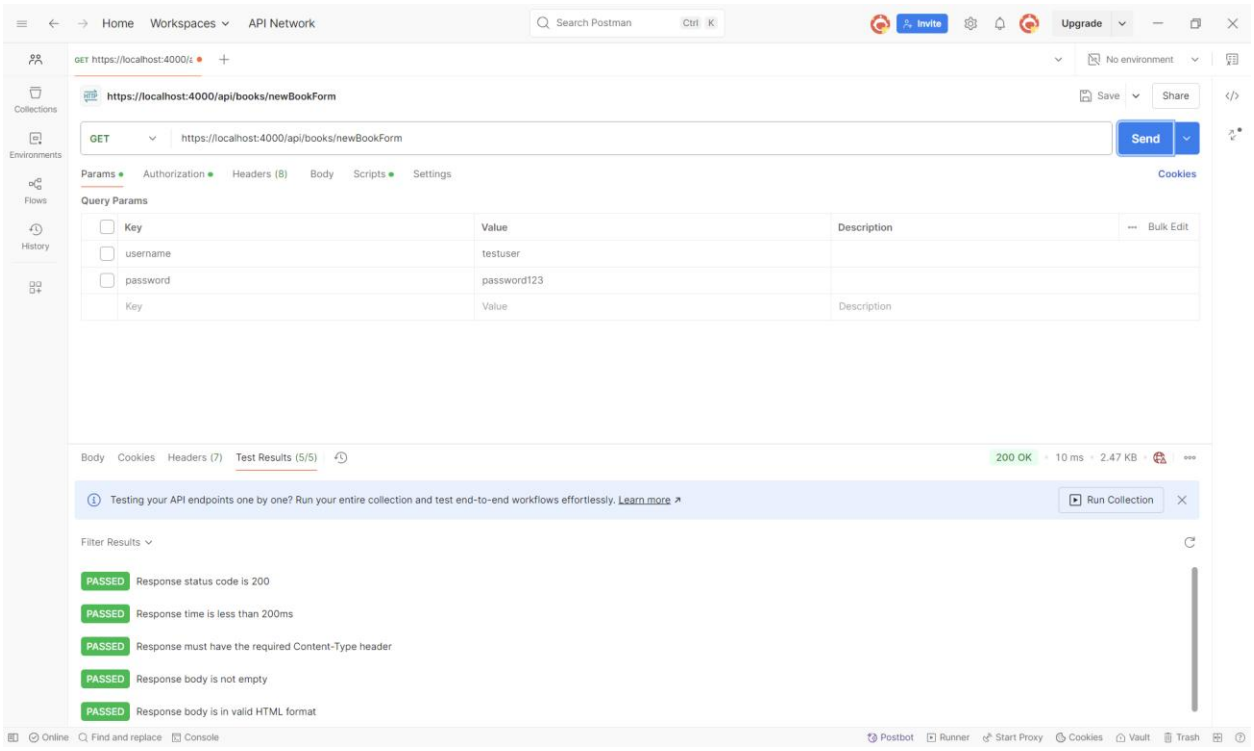
c. Protected Resource



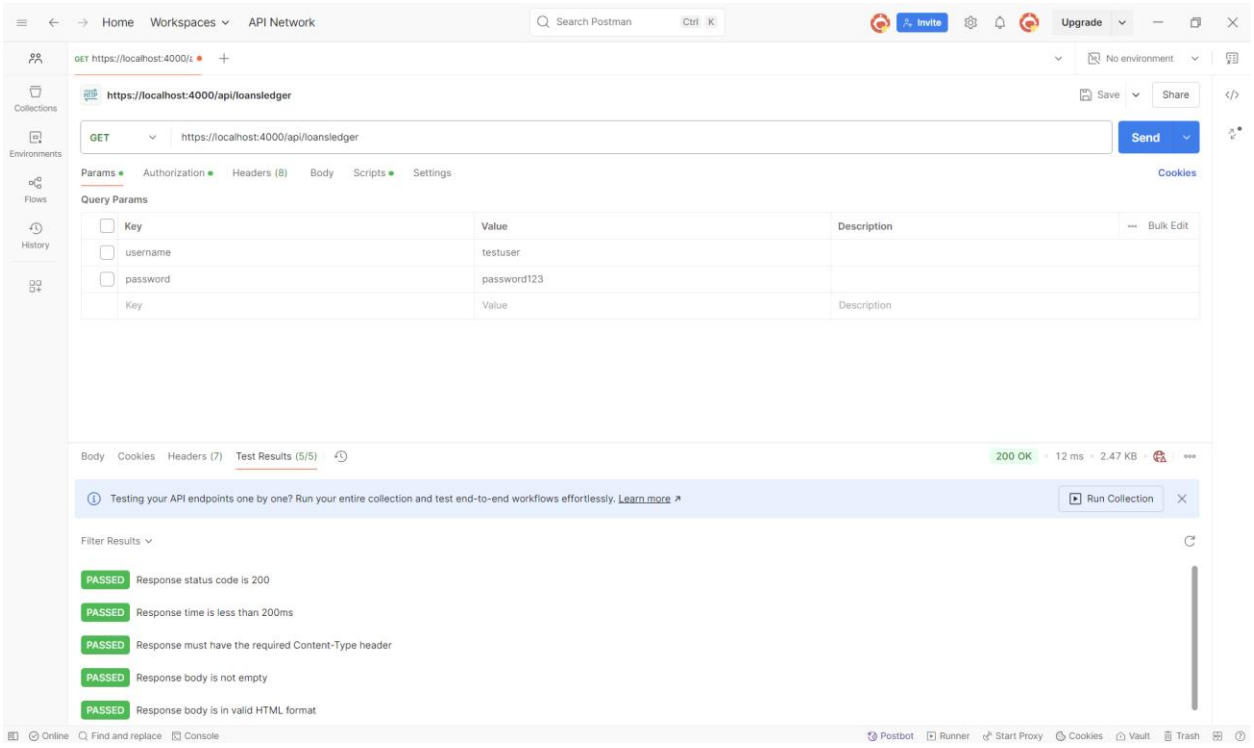
d. View All Books



e. Add Book Exchange



f. Update Delete Exchange



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 Middlware.