# 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.

*A screenshot of a computer

AI-generated content may be incorrect.*

*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  
*A screenshot of a computer

AI-generated content may be incorrect.*  
*Figure 3: Custom Middleware Response*

*A screenshot of a computer

AI-generated content may be incorrect.*

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

*A screenshot of a computer

AI-generated content may be incorrect.*

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

*A screenshot of a computer

AI-generated content may be incorrect.*

*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.*

*A screenshot of a computer

AI-generated content may be incorrect.*

*Figure 7: Logging in (Auth Middleware)*

*A screenshot of a computer

AI-generated content may be incorrect.Figure 8: Error handling Middeware 404*

*A screenshot of a computer

AI-generated content may be incorrect.*

*Figure 9: Error handling middleware*

*Method: Attempting to use* [*http://localhost:4000/non-existent-route*](http://localhost:4000/non-existent-route)

*Figure 10: Of the project directory structure showcasing MVC organization.*A screenshot of a computer

AI-generated content may be incorrect.*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.*A screenshot of a computer

AI-generated content may be incorrect. *Note: owner exists already when created a new book as shown in bookController.js*

*Figure 12: Of the bookRoutes.js file with defined routes.  
A screenshot of a computer

AI-generated content may be incorrect.*

*Figure 13: Of the home page as displayed in the browser.  
A screenshot of a computer

AI-generated content may be incorrect.*

*Figure 14: Of all personalized user interface pages.  
A screenshot of a computer

AI-generated content may be incorrect.  
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.*A screenshot of a computer

AI-generated content may be incorrect.

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

* 1. *Register / Sign up*

*A screenshot of a computer

AI-generated content may be incorrect.*

* 1. *Login*

*A screenshot of a computer

AI-generated content may be incorrect.*

* 1. *Protected Resource*

*A screenshot of a computer

AI-generated content may be incorrect.*

* 1. *View All Books*

*A screenshot of a computer

AI-generated content may be incorrect.*

* 1. *Add Book Exchange*

*A screenshot of a computer

AI-generated content may be incorrect.*

* 1. *Update Delete Exchange*

*A screenshot of a computer

AI-generated content may be incorrect.*

*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.*

*A screenshot of a computer

AI-generated content may be incorrect.*