# Module 8: Final Project

# Brandon Trinkle

# IFT 458: Middleware Programming

# Professor Dinesh Sthapit

# April 21, 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 scalable Book Exchange service.

**Scope and Limitations**

* *Scope*: Modules 5–8 cover advanced routing and middleware (Module 6), SSL and HTTPS server configuration (Module 7), error handling and status codes (Module 7 Activity 2), and container deployment (Module 6 Activity 2).
* *Limitations*: Module 8 content was not available at submission; screenshots and detailed reflections for Module 8 will be added upon receipt of materials.

**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*

Screenshots from Lab 7   
*A screenshot of a computer

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

1. *Screenshot 1: Of the terminal showing successful installation of dependencies.*A screenshot of a computer

   AI-generated content may be incorrect.
2. *Screenshot 2: 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.*
3. *Screenshot 3: Of the BookModel.js file with a brief explanation.*A screenshot of a computer program

   AI-generated content may be incorrect. *Note: BookModel.js file defines a Mongoose schema named bookExchangeSchema that establishes the structure for storing book exchange data in MongoDB. Each book document includes required fields such as title, author, and description, ensuring that all necessary details are provided. Regarding the question in this section; we already store userID of the user who added the book with ‘owner’.*
4. *Screenshot 4: Of the bookController.js with the implemented 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*
5. *Screenshot 5: Of the bookRoutes.js file with defined routes.  
   A screenshot of a computer

   AI-generated content may be incorrect.*
6. *Screenshot 6: Of updated view files (\*.ejs) and changes made.  
   A screenshot of a computer

   AI-generated content may be incorrect.  
   Note: I removed the Genre section as its not defined in our model.*
7. *Screenshot 7: Of the home page as displayed in the browser.  
   A screenshot of a computer

   AI-generated content may be incorrect.*
8. *Screenshot 8: 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*
9. *Screenshot 9: Of the testing process and responses for different API endpoints.*A screenshot of a computer

   AI-generated content may be incorrect.
10. *Screenshot 10: 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.*

*A screenshot of a computer

AI-generated content may be incorrect.*