Module 8: Final Project

Submitted By: Praveen Kumar Siddela

IFT 458 & IFT 554: Middleware Programming & Database Security

Dinesh Sthapit

Due Date: 04/26/2024

Introduction

Overview of the project

The project's objective is to create a Book Exchange Application facilitating users to list and exchange books seamlessly with their peers. It's designed as a lightweight web application prioritizing smooth user experiences for registration, password management, and book listing. Following the MVC (Model-View-Controller) pattern in its architecture ensures scalability and maintainability by separating concerns effectively.

Problem Statement

Conventional book exchange methods are inefficient and inaccessible, often relying on physical meetings or complex online platforms. This initiative aims to overcome these shortcomings by offering a digital platform where users can effortlessly list their books, explore available titles, and initiate exchanges with others. The aim is to simplify the book exchange process, making it convenient and accessible to a broader audience.

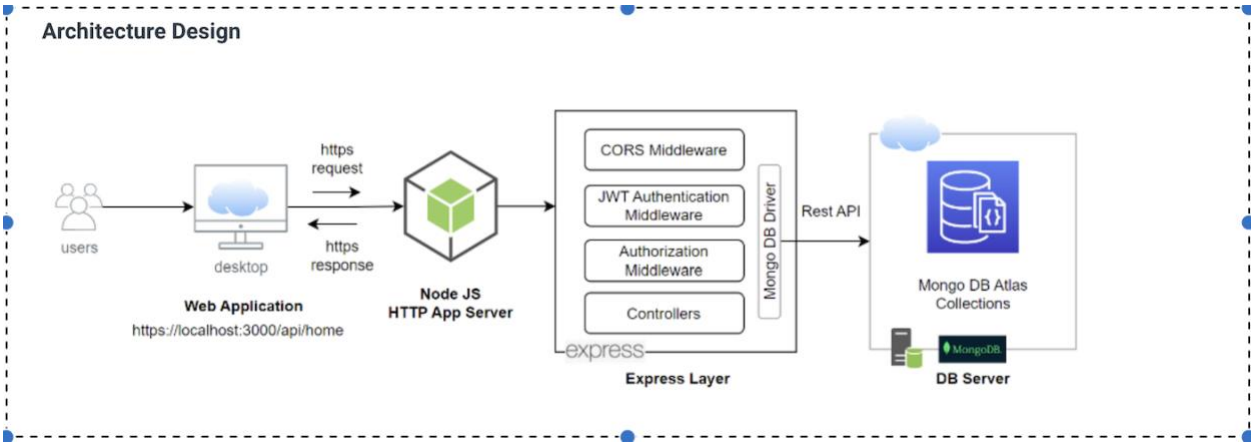
Scope and Limitations

The project encompasses developing a web-based application featuring user authentication, book listing, and exchange functionalities. Users can register, log in, list their books, browse titles, and initiate exchanges. However, the application doesn't handle physical book shipments or payments; it solely facilitates the exchange process by connecting users based on their preferences.

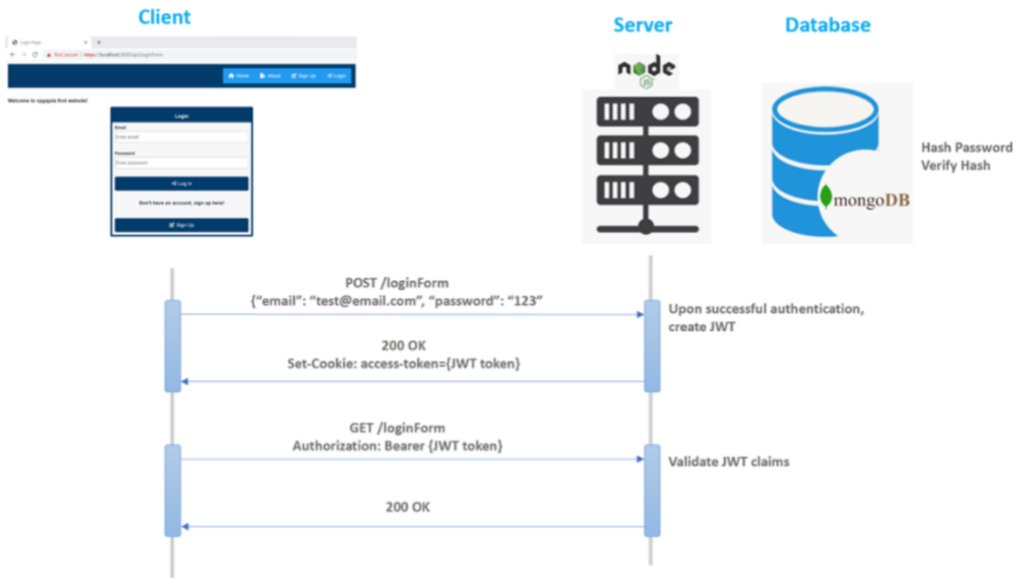
Justification of the Technology Stack Used

The technology stack comprises Node.js for server-side development, Express.js for constructing the web application framework, MongoDB for efficient data storage, and EJS (Embedded JavaScript) for server-side templating. These technologies were chosen for their reliability, scalability, and user-friendly nature in web application development. Node.js and Express.js offer a lightweight and effective server environment, while MongoDB ensures flexibility and scalability in storing user and book data. EJS enables dynamic content rendering on the server side, enhancing user interaction.

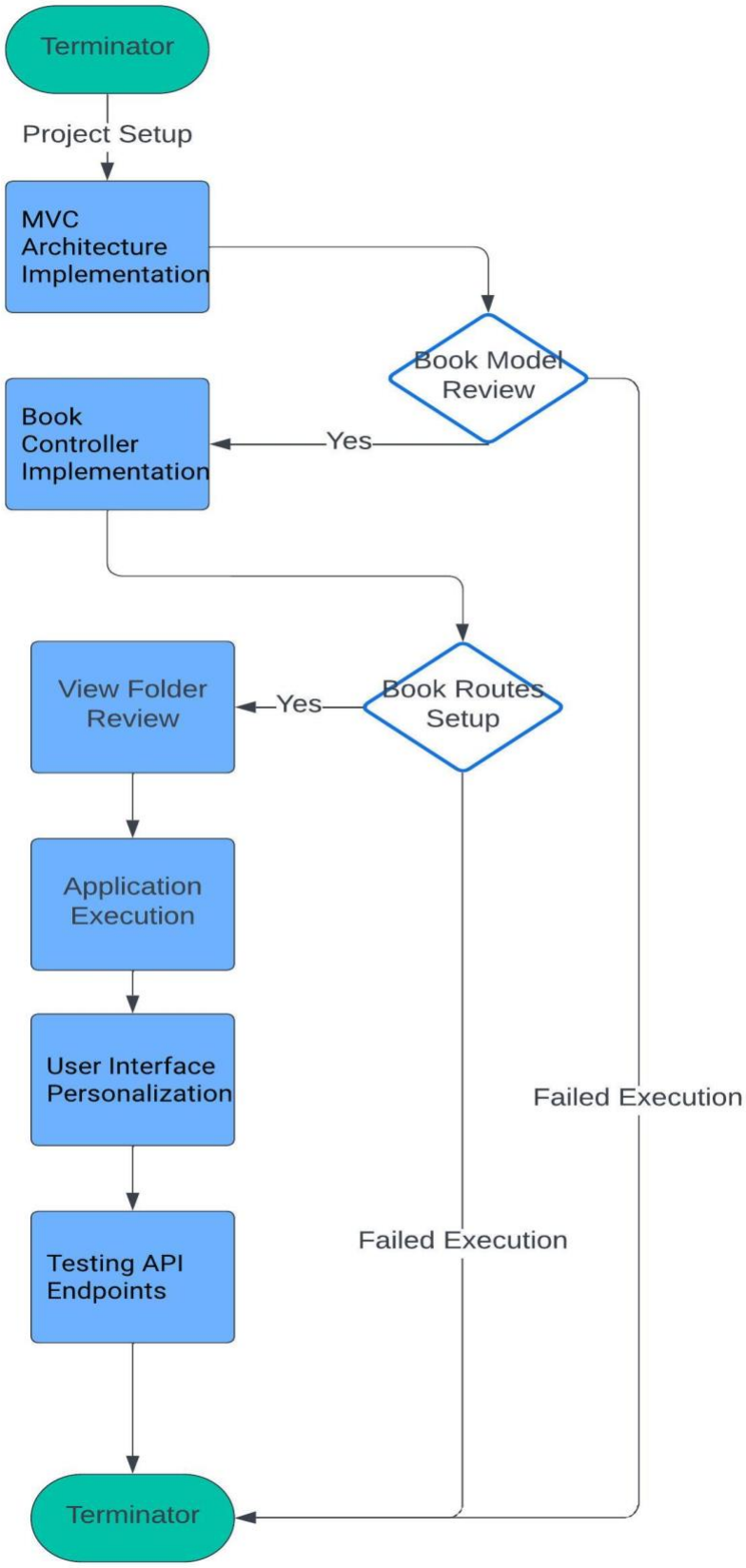
Architecture Diagrams



Security Flow:



Flowchart


Terminator
Project Setup
MVC Architecture Implementation
Book Model Review
Yes
Book Controller Implementation
Book Routes Setup
Yes
View Folder Review
Application Execution
User Interface Personalization
Testing API Endpoints
Failed Execution
Failed Execution
Terminator

Pseudocode:

// Server-side Process:

1. Respond to incoming requests along specified routes.
2. Determine the appropriate controller function for each request.
3. Handle data processing and interaction with the model.
4. Formulate and dispatch responses back to the client.

// Model (Data Management):

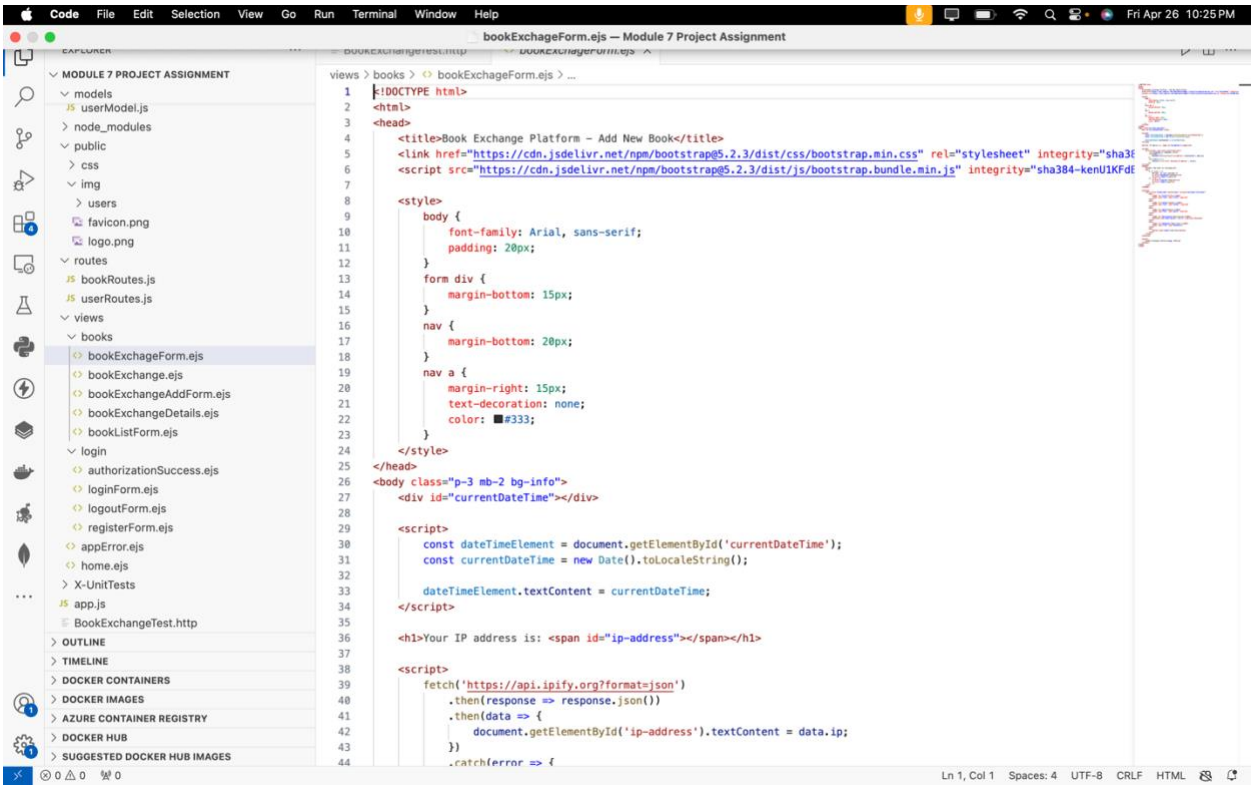
1. Establish structured schemas for book and user data representation.
2. Develop methods to execute database operations, including creation, retrieval, updating, and deletion.
3. Implement procedures for data validation and manipulation.
4. Provide processed data to the controller for further handling.

// Client-side Actions:

1. Dynamically construct user interface components based on application state.
2. Monitor and respond to user interactions, like form submissions and button clicks.
3. Initiate relevant actions, such as sending requests to the server, upon user interaction.
4. Reflect updates to the interface in accordance with server responses.

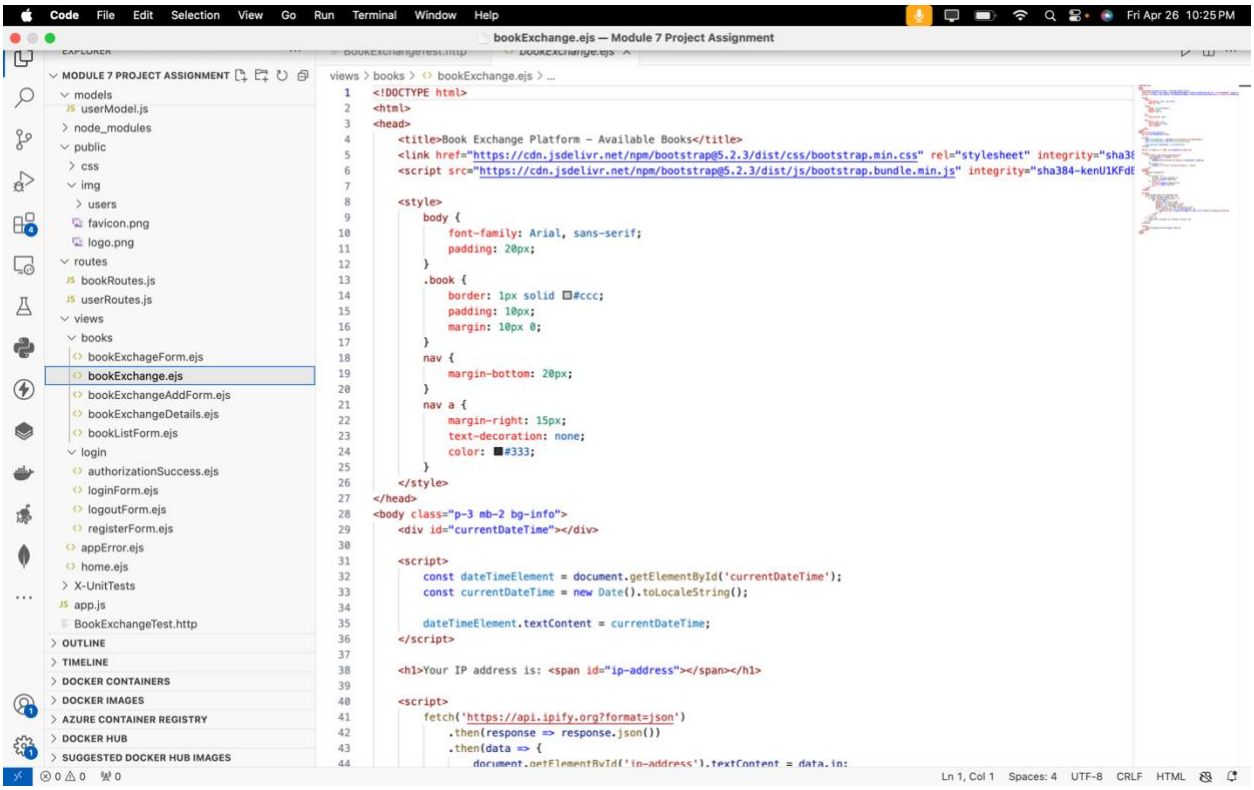
Code Snippets:

bookExchangeForm.ejs



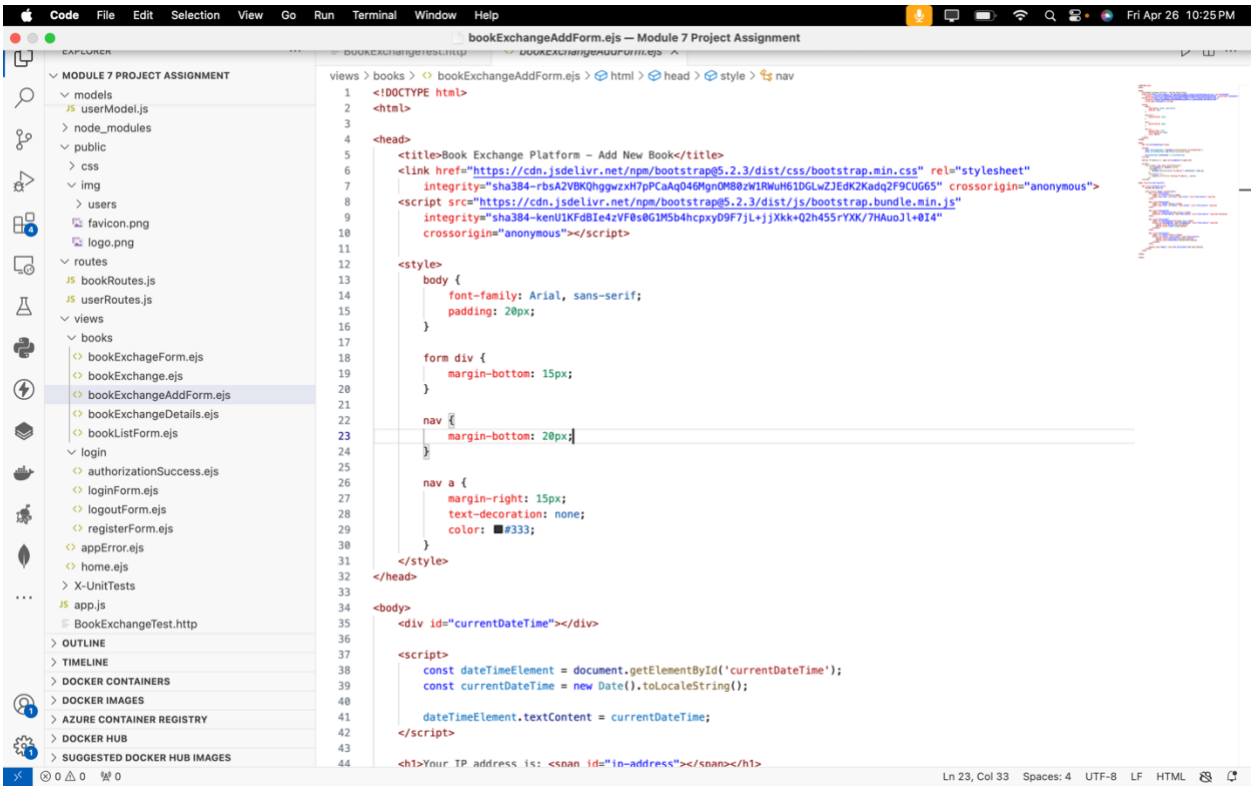
1. **Form Submission:** The <form> element defines a form for submitting book details to the server. It specifies the action attribute as "/books/add" for sending the form data to the appropriate endpoint on the server.
2. **Input Fields:** Various <input> elements are used to capture information about the book, such as title, author, genre, and description. The required attribute ensures that these fields must be filled out before the form can be submitted.
3. **File Upload:** An <input type="file"> element allows users to upload a book cover image along with the other details.

bookExchange.ejs



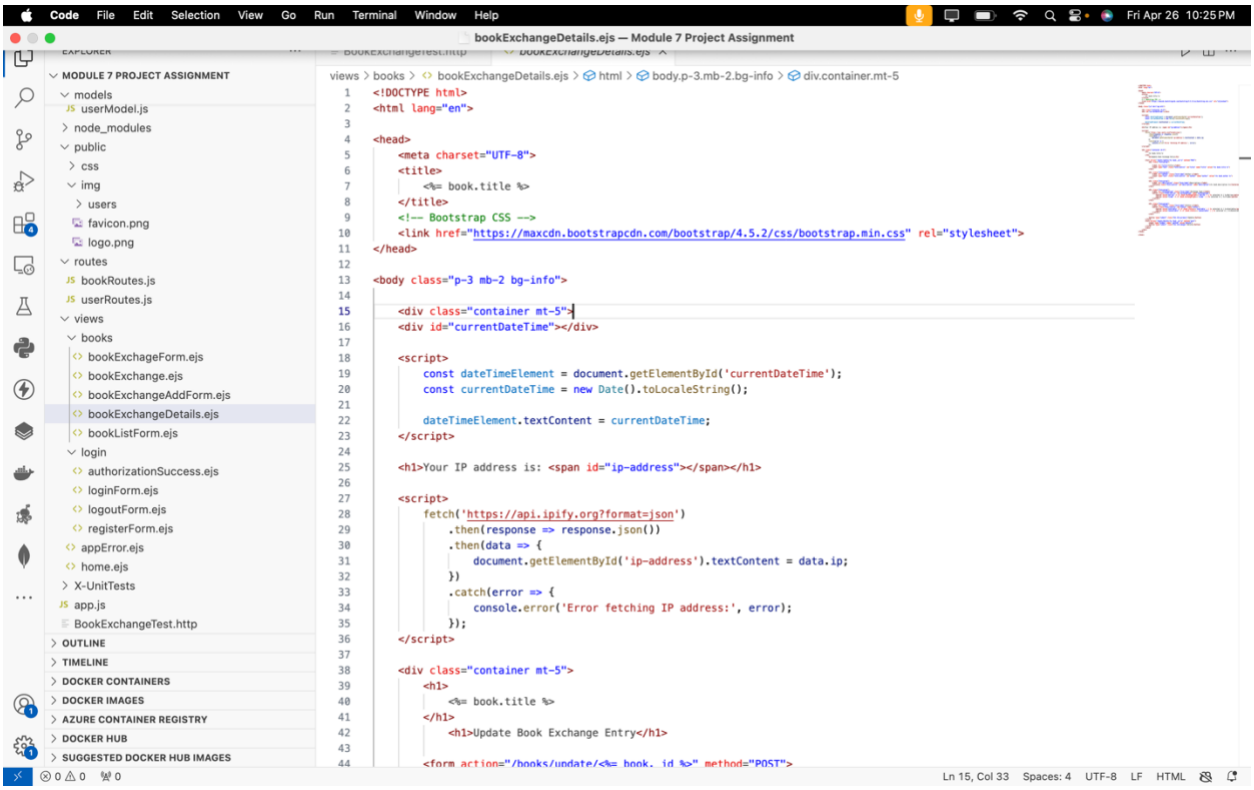
1. **Error Handling:** The Fetch API is used to asynchronously fetch data from an external API (https://api.ipify.org?format=json). Error handling is implemented to catch and log any errors that may occur during the fetch operation.
2. **Footer:** The <footer> section provides copyright information for the Book Exchange Platform, ensuring proper attribution and ownership acknowledgment.

bookExchangeAddForm.ejs

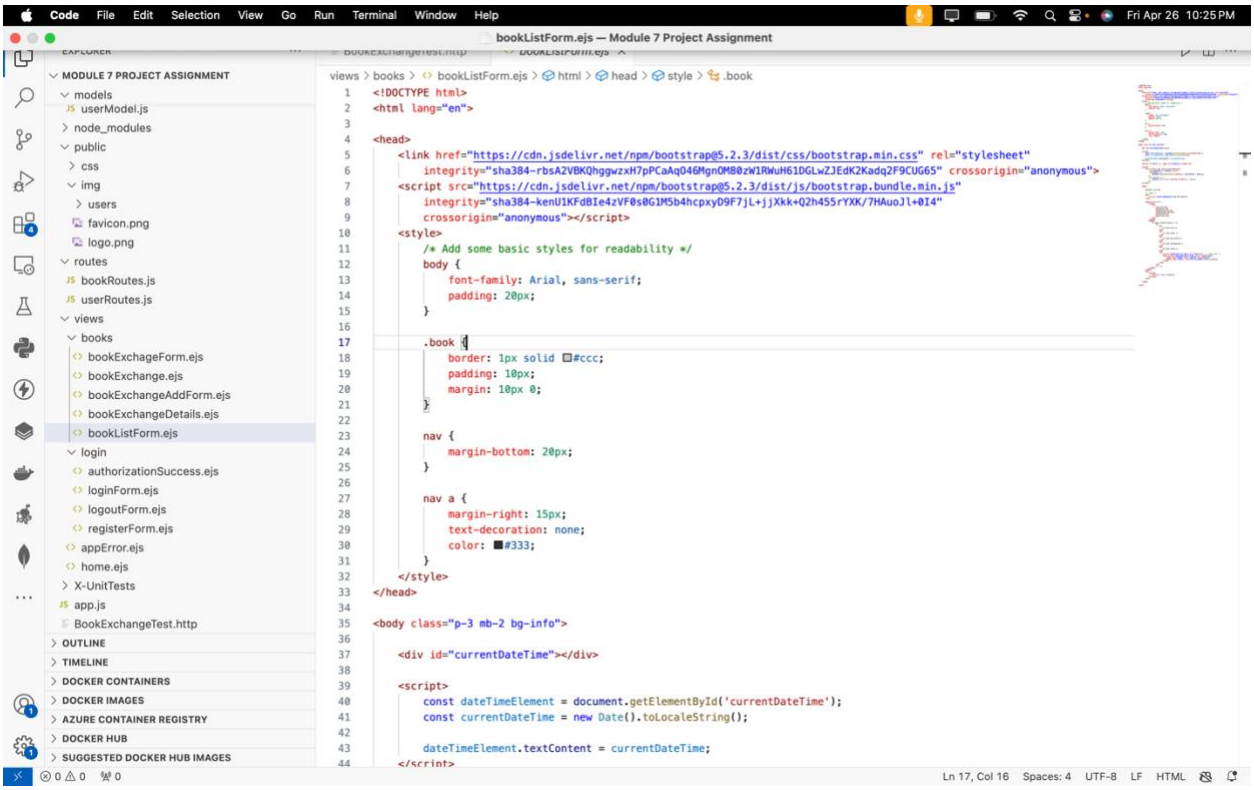


1. **External Dependencies:** This is achieved through the <link> and <script> tags in the <head> section.
2. **Dynamic Content:** JavaScript embedded within the HTML document retrieves and displays the current date and time, as well as the user's IP address, using the Fetch API. This dynamic content is injected into specific elements within the <body> section.
3. **Form Submission:** The <form> element defines a form for adding a new book. It specifies the action attribute as "/books" and the method attribute as "post", indicating that form data will be sent to the "/books" endpoint using the HTTP POST method upon submission.

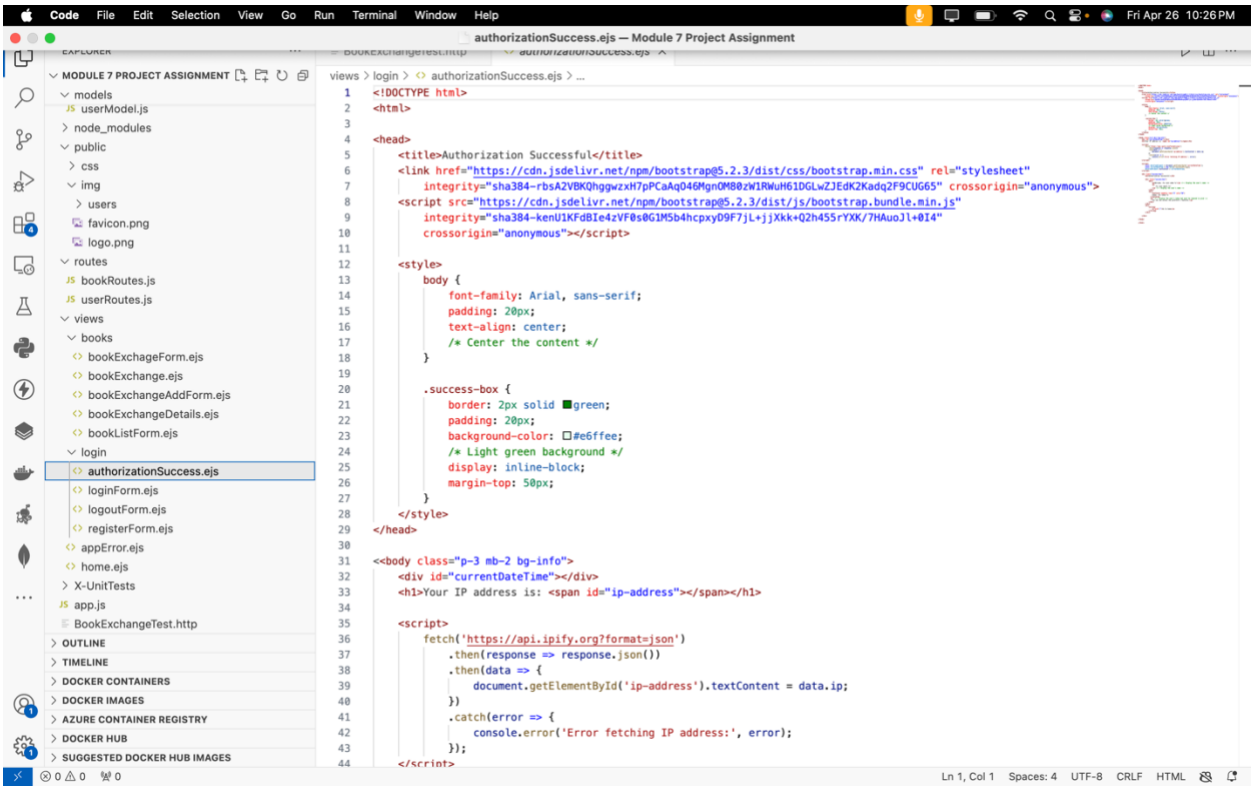
bookExchangeDetails.ejs



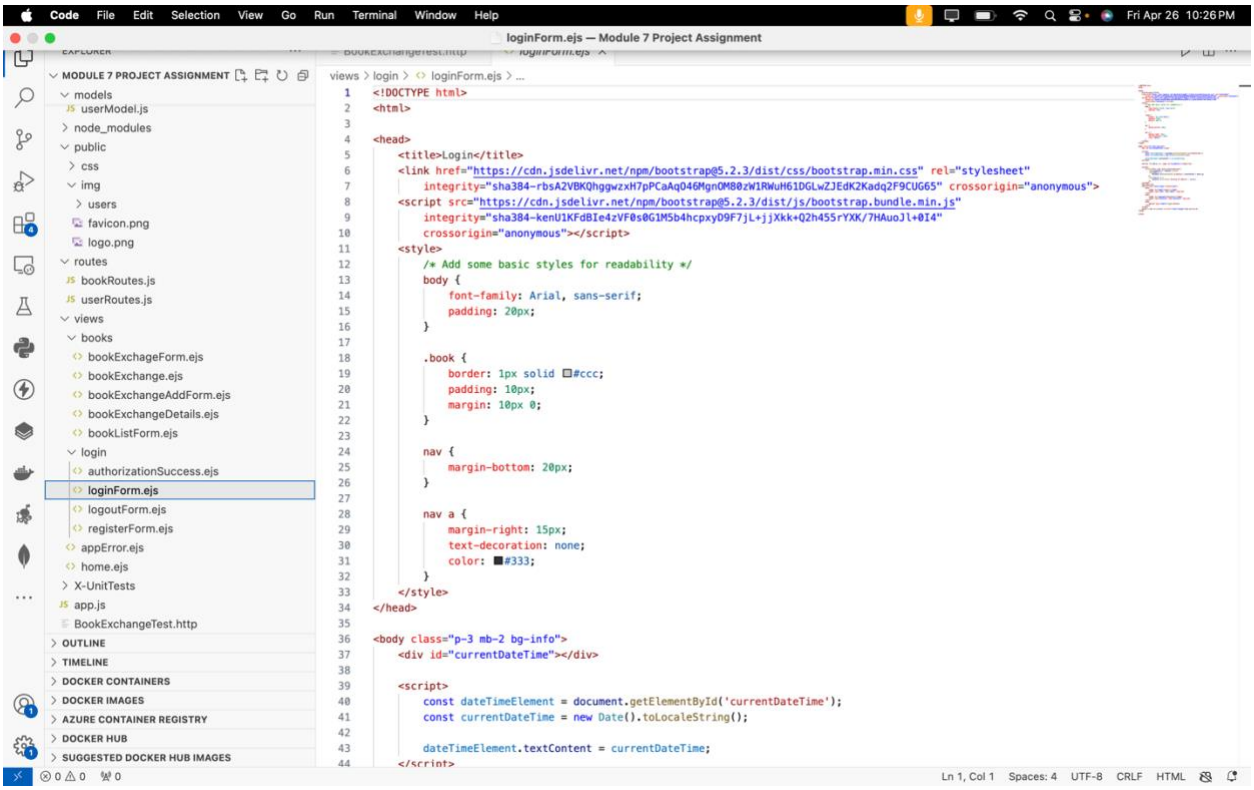
bookListForm.ejs



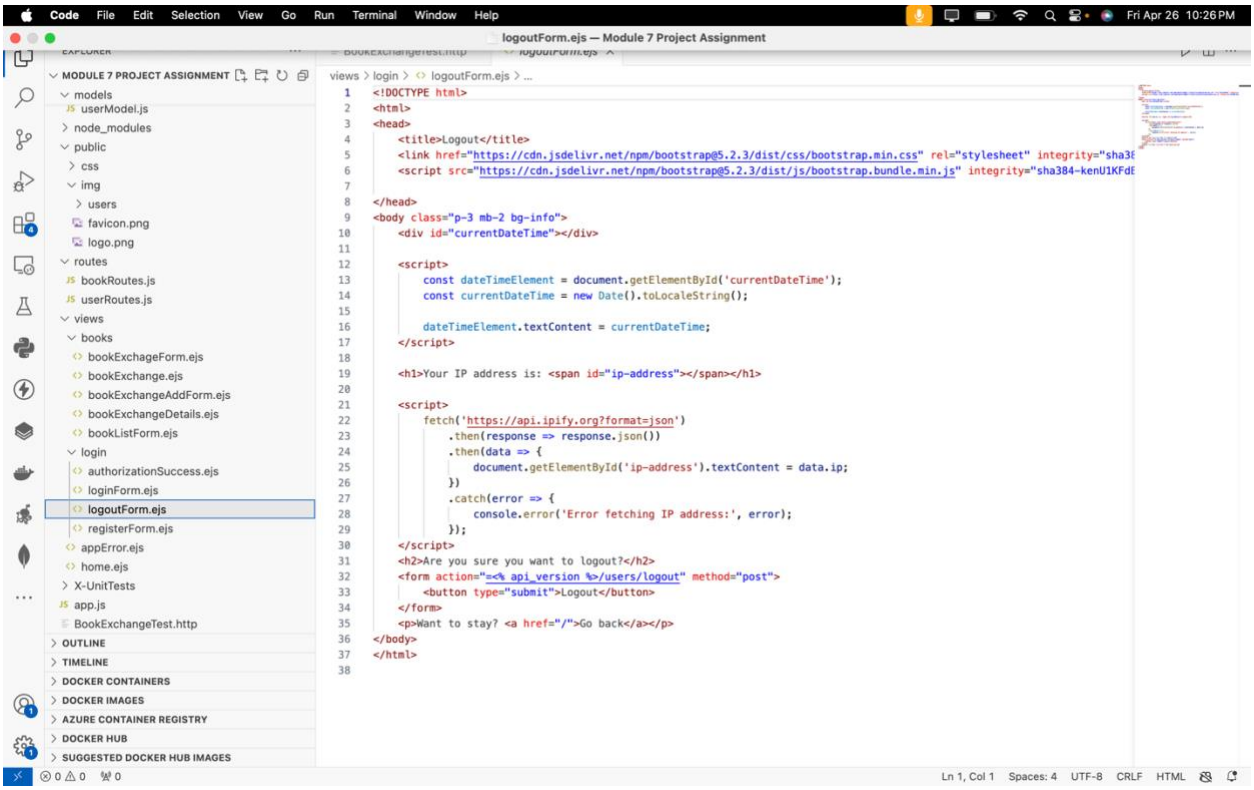
authorizationSuccess.ejs



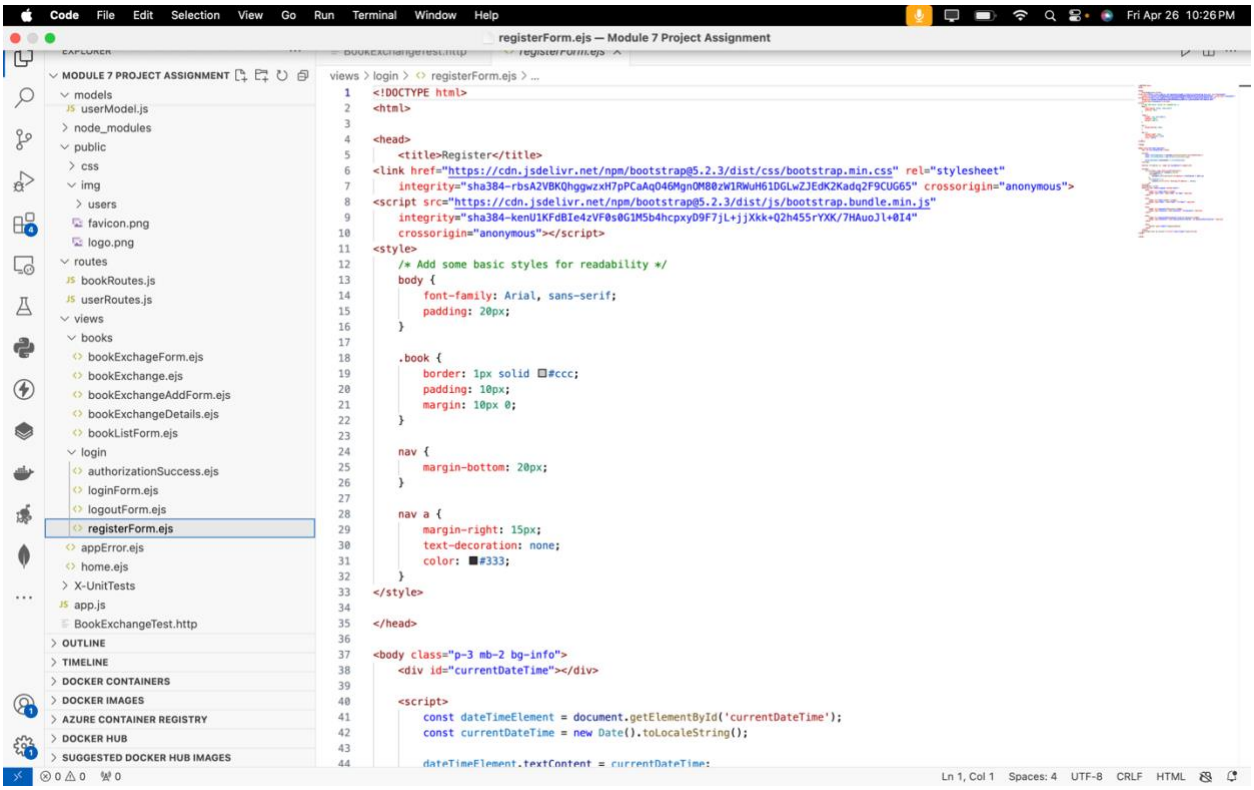
loginForm.ejs



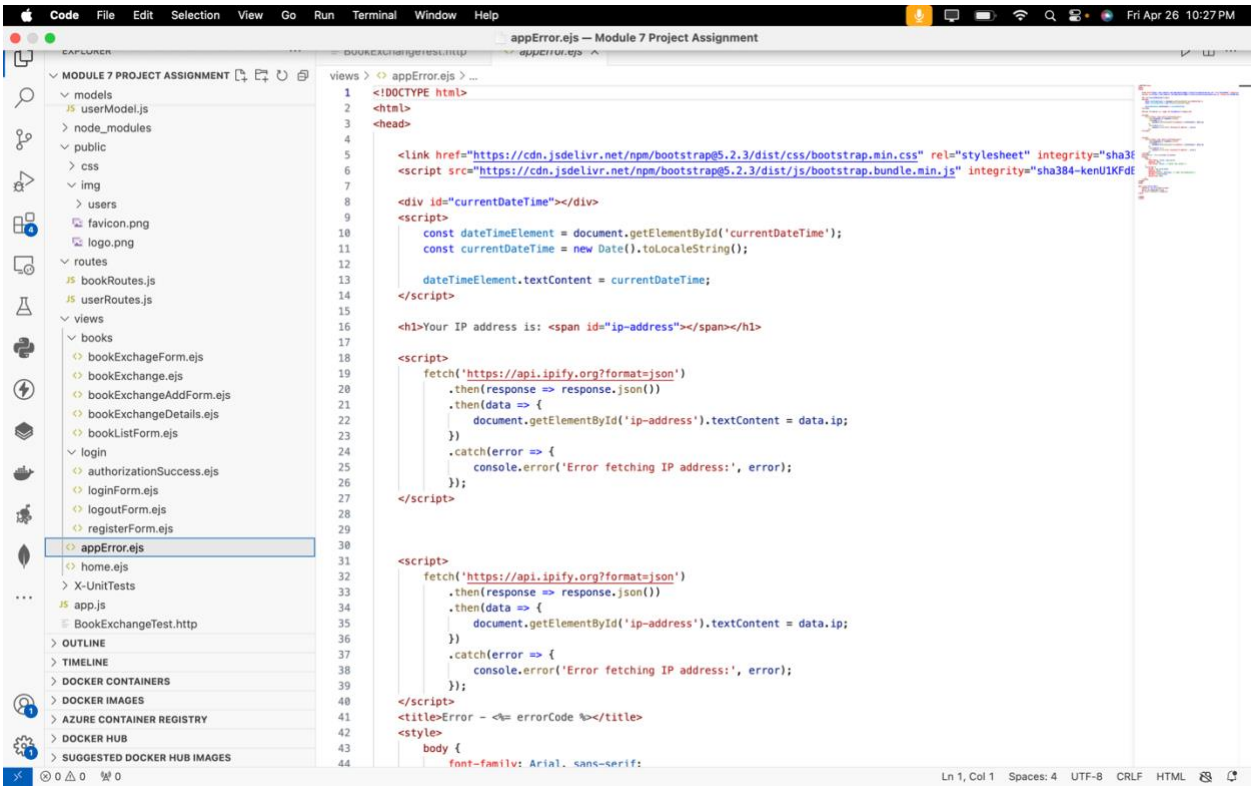
logoutForm.ejs



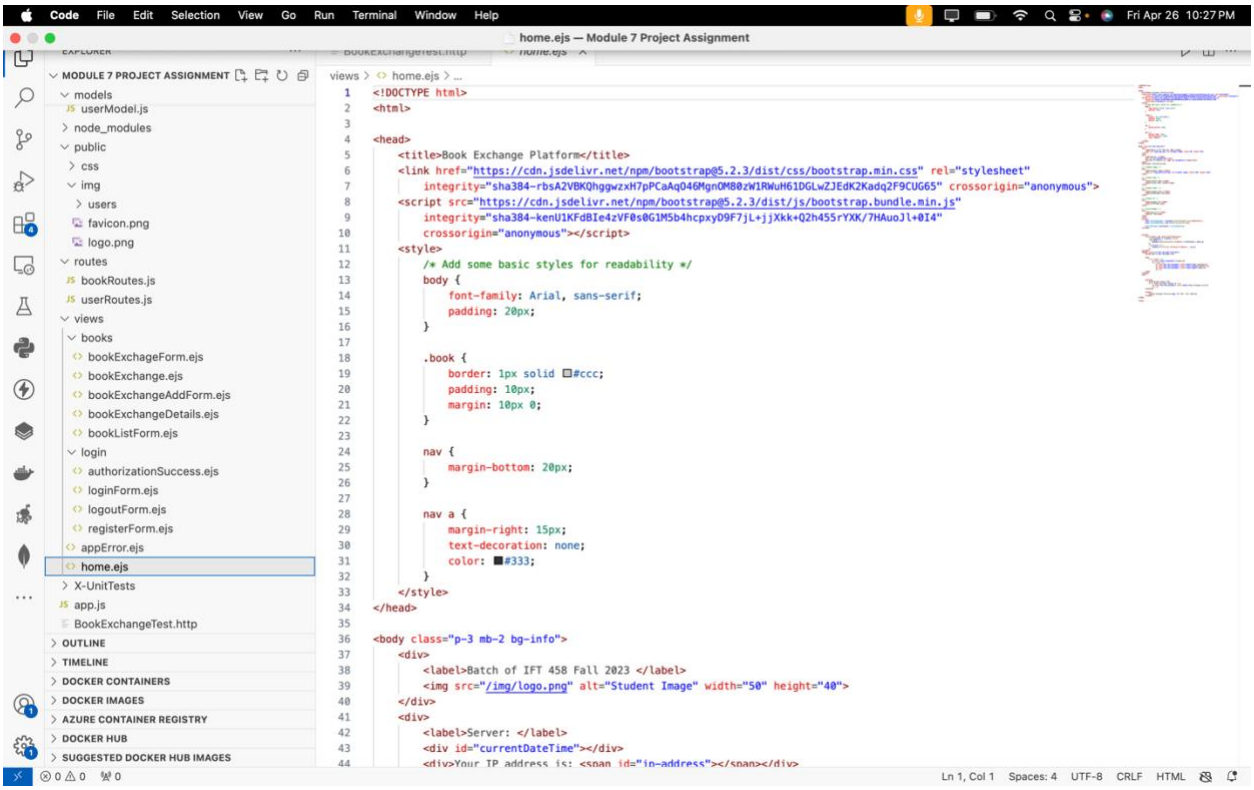
registerForm.ejs



appError.ejs



home.ejs

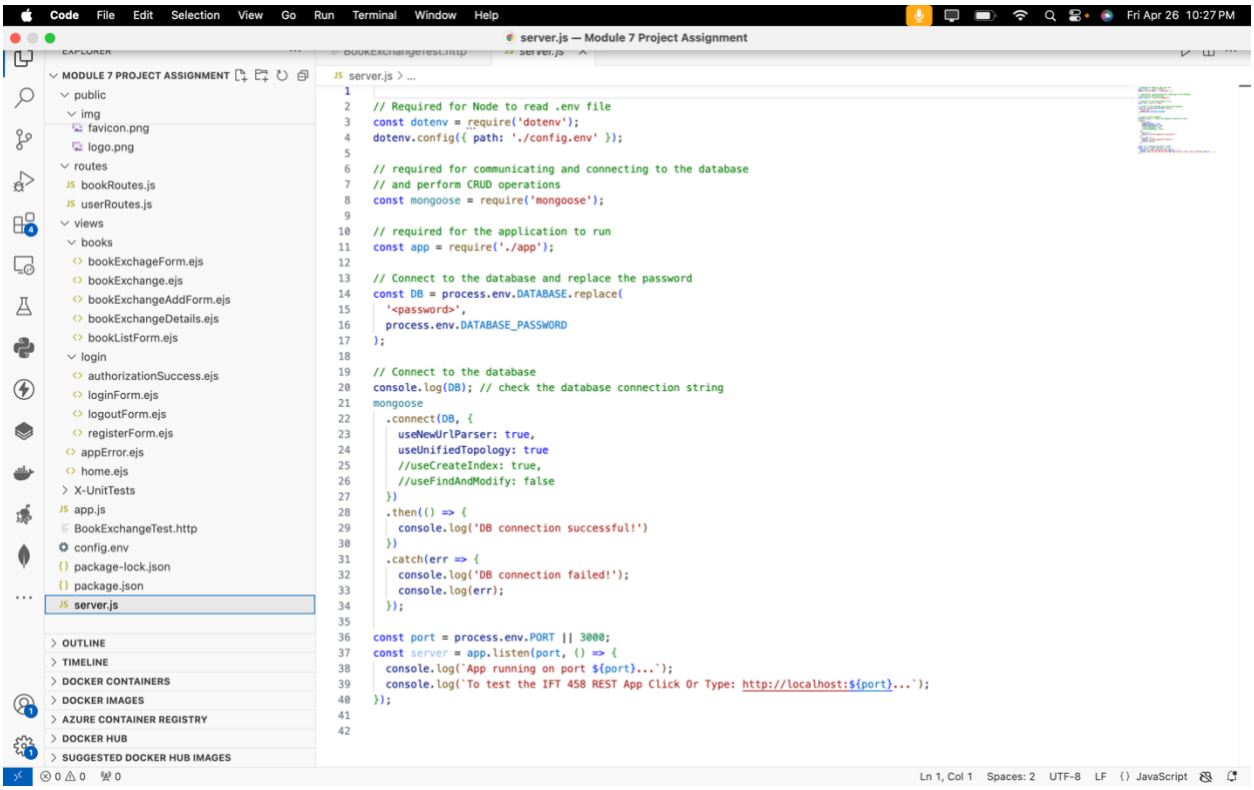

Code File Edit Selection View Go Run Terminal Window Help
Fri Apr 26 10:27 PM
home.ejs — Module 7 Project Assignment
MODULE 7 PROJECT ASSIGNMENT
models
userModel.js
node_modules
public
css
img
users
favicon.png
logo.png
routes
bookRoutes.js
userRoutes.js
views
books
bookExchageForm.ejs
bookExchange.ejs
bookExchangeAddForm.ejs
bookExchangeDetails.ejs
bookListForm.ejs
login
authorizationSuccess.ejs
loginForm.ejs
logoutForm.ejs
registerForm.ejs
appError.ejs
home.ejs
X-UnitTests
app.js
BookExchangeTest.http
OUTLINE
TIMELINE
DOCKER CONTAINERS
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMAGES
views > home.ejs > ...
<!DOCTYPE html>
<html>
<head>
<title>Book Exchange Platform</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4"
crossorigin="anonymous"></script>
<style>
/* Add some basic styles for readability */
body {
font-family: Arial, sans-serif;
padding: 20px;
}
.book {
border: 1px solid #ccc;
padding: 10px;
margin: 10px 0;
}
nav {
margin-bottom: 20px;
}
nav a {
margin-right: 15px;
text-decoration: none;
color: #333;
}
</style>
</head>
<body class="p-3 mb-2 bg-info">

<label>Batch of IFT 458 Fall 2023 </label>
<img src="/img/logo.png" alt="Student Image" width="50" height="40">

<label>Server: </label>

Your IP address is:
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF HTML

server.js



API Specifications:

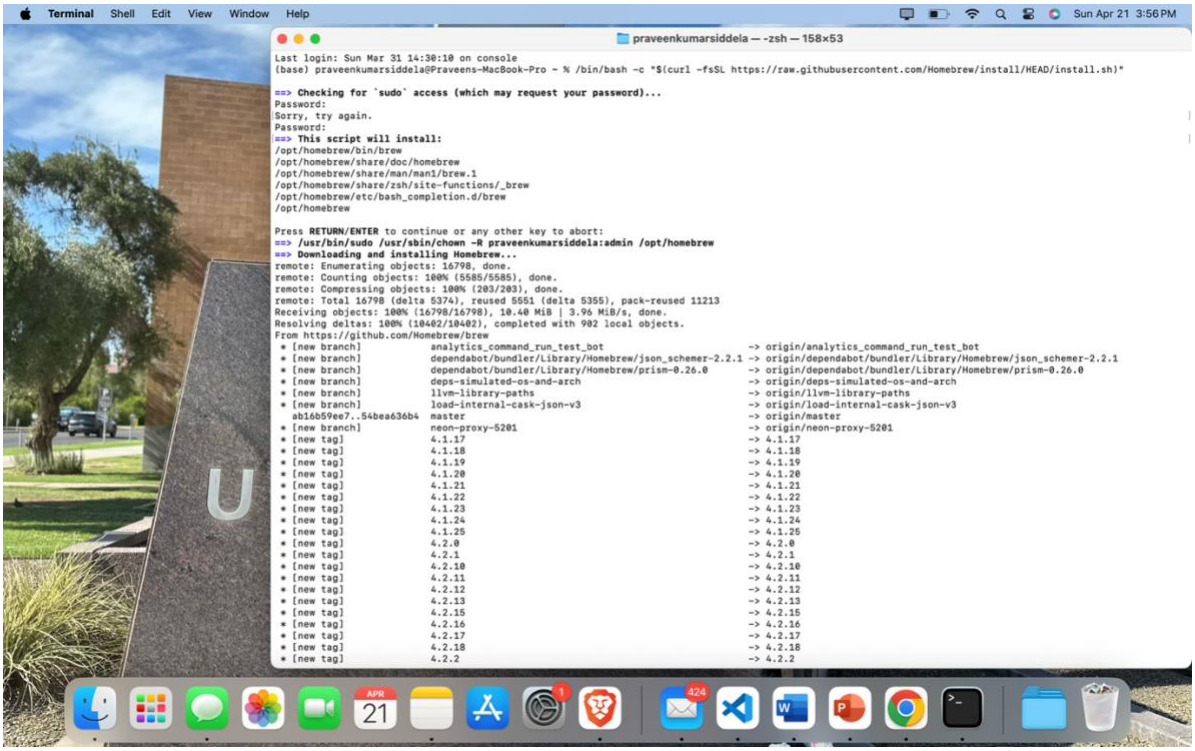
The API specifications detail the endpoints, request methods, and expected formats necessary for communicating with the backend of the application. These specifications follow RESTful principles and encompass functionalities like user authentication, book listing, and exchange.

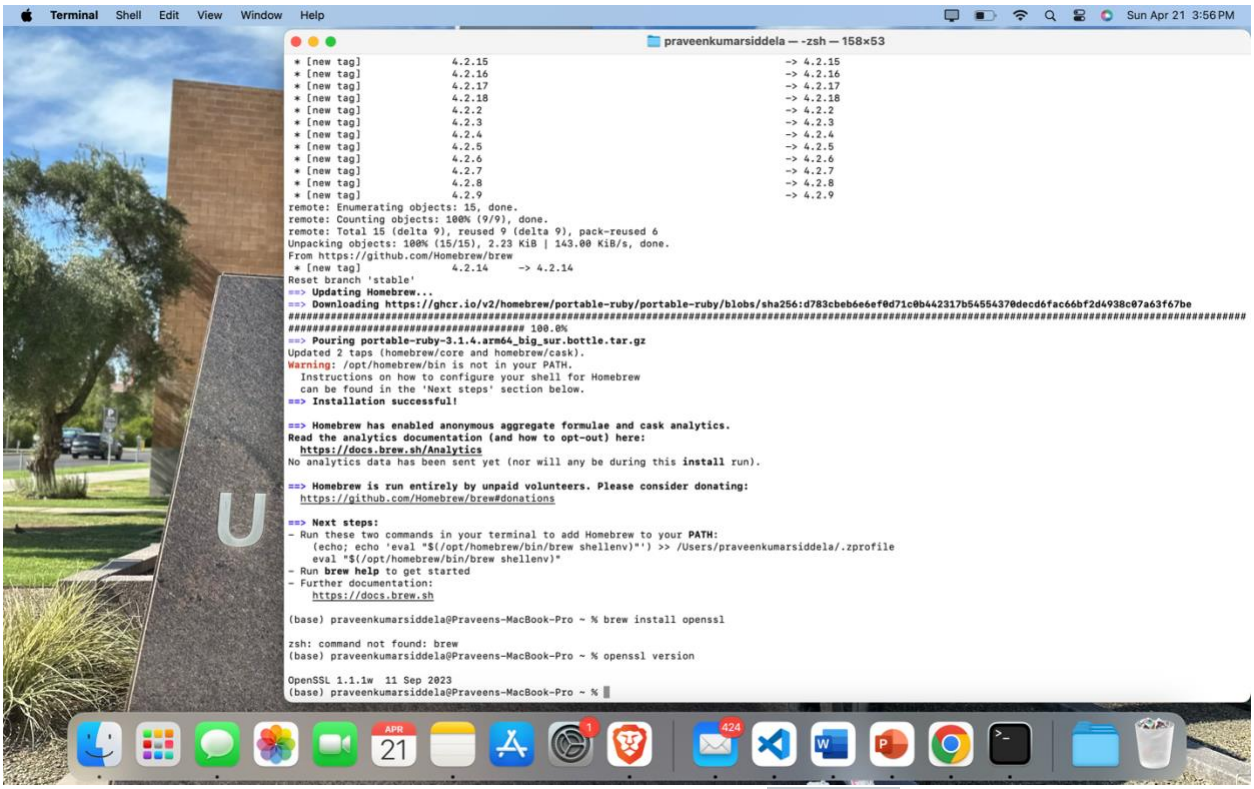
Specific endpoints include:

- Registering or signing up, accessed via the POST method at https://localhost:4000/users/signup
- Logging in, utilizing the POST method at https://localhost:4000/users/login
- Accessing protected resources, achieved through the GET method at https://localhost:4000/books
- Viewing all available books, accessible via the GET method at https://localhost:4000/books
- Adding a book for exchange, done through the POST method at https://localhost:4000/books/newBookForm
- Updating or deleting a book exchange, facilitated via the POST method at http://localhost:4000/books/edit/65e565c188ab31ac49627500

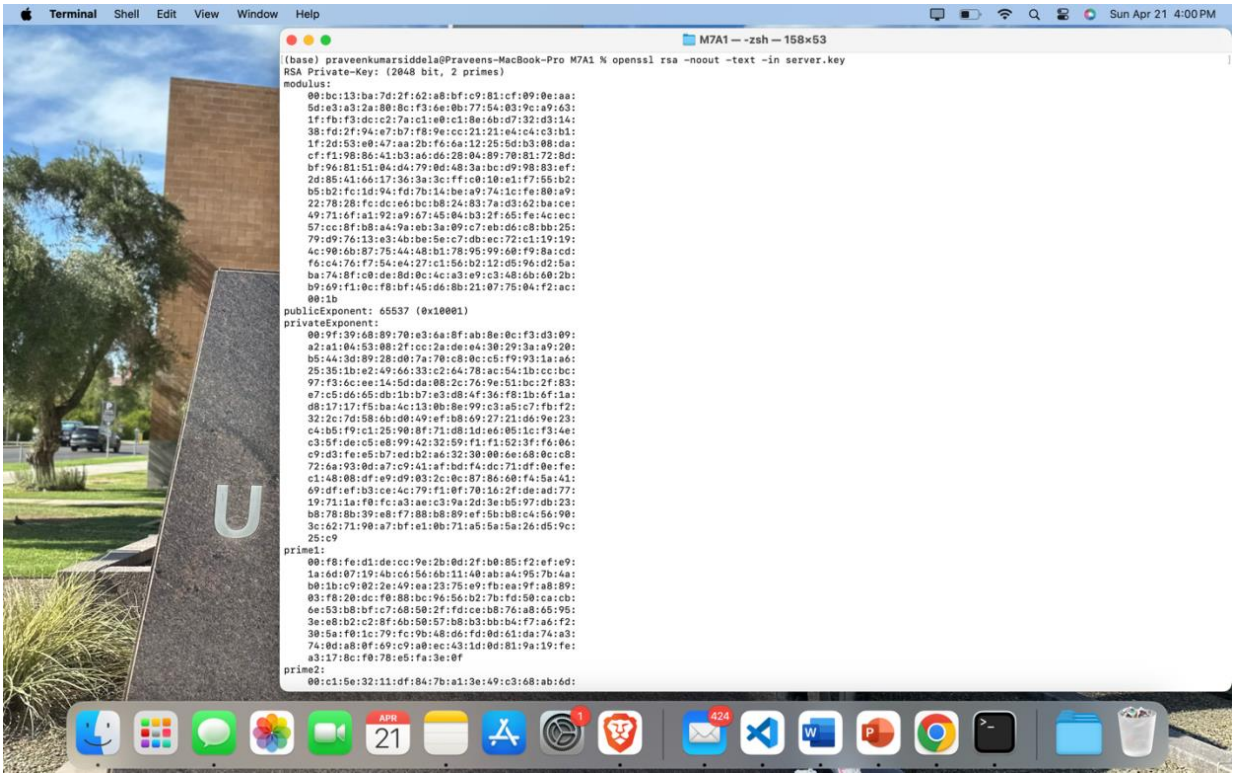
Module 7 Activity 1: Generating SSL Certificate and Configuring HTTPS Server in Node.js on Windows

Screenshot 1: Showing the OpenSSL installed confirmation.

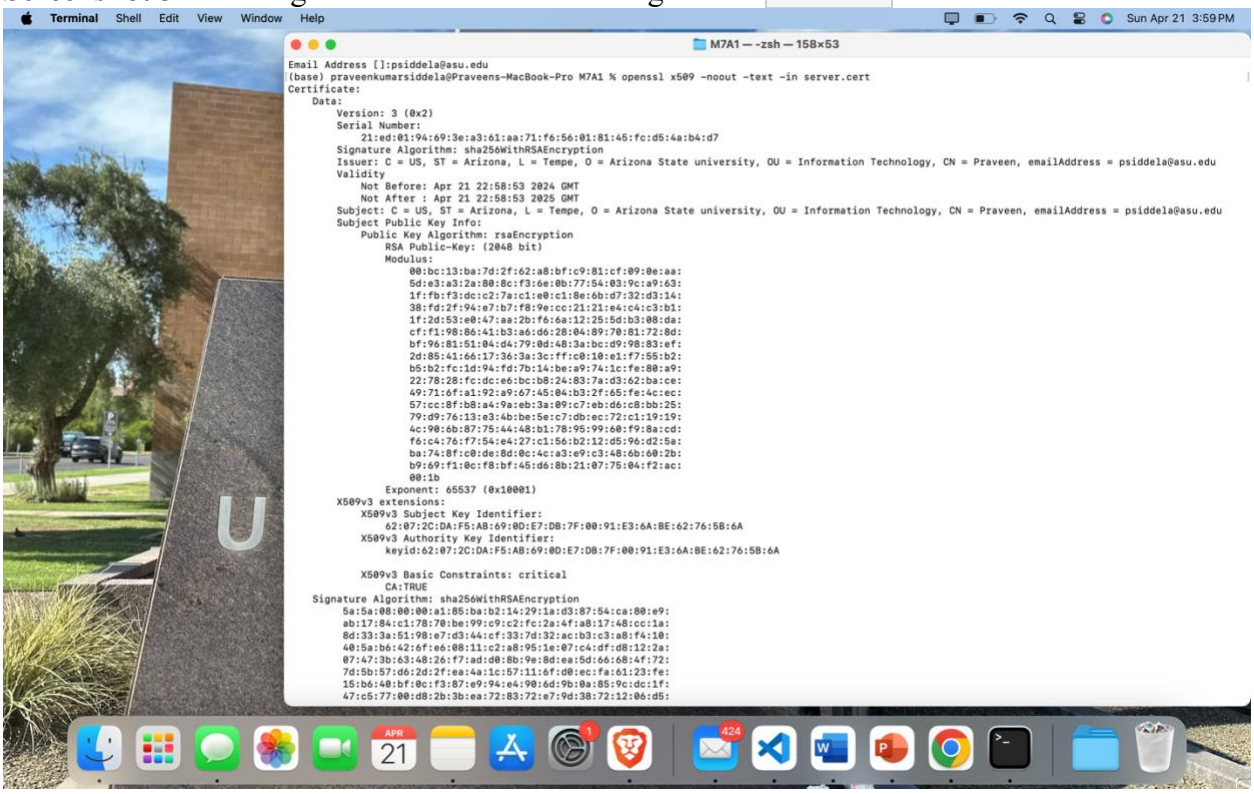




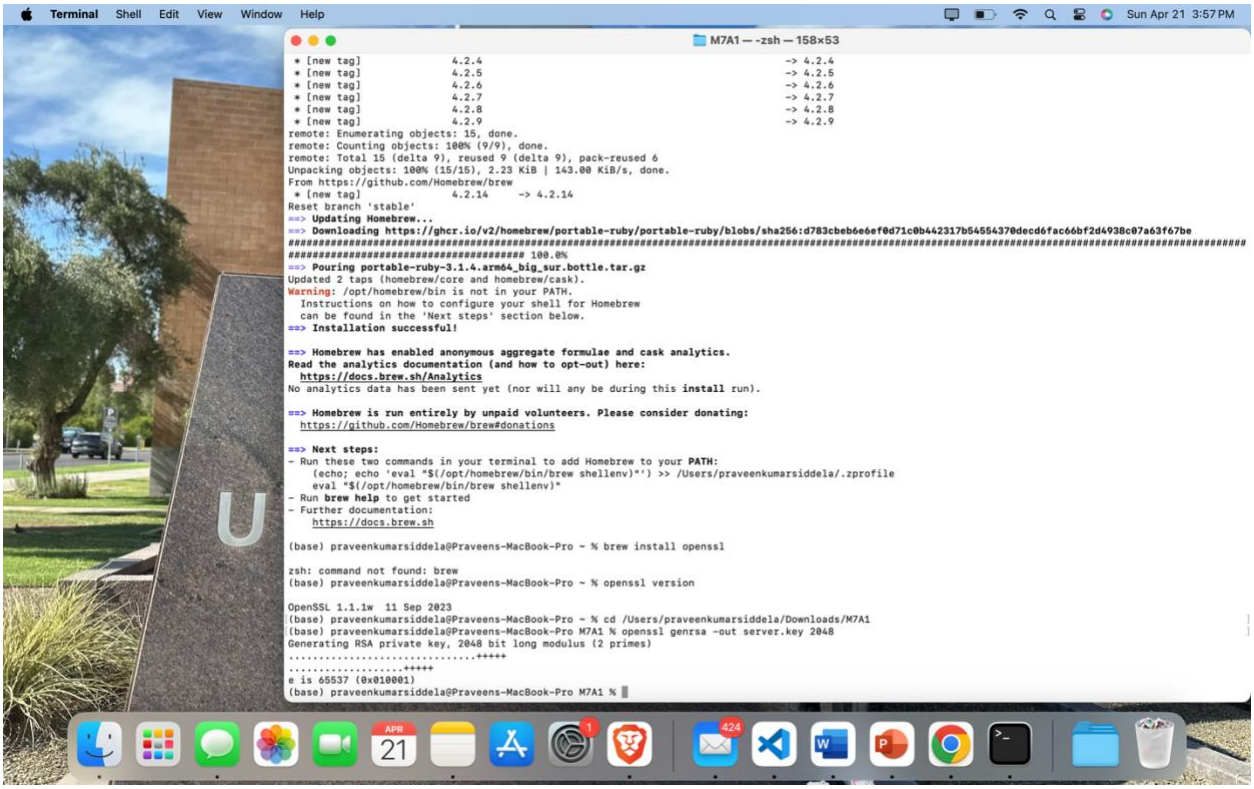
Screenshot 2: Showing Command Prompt after generating server.key.

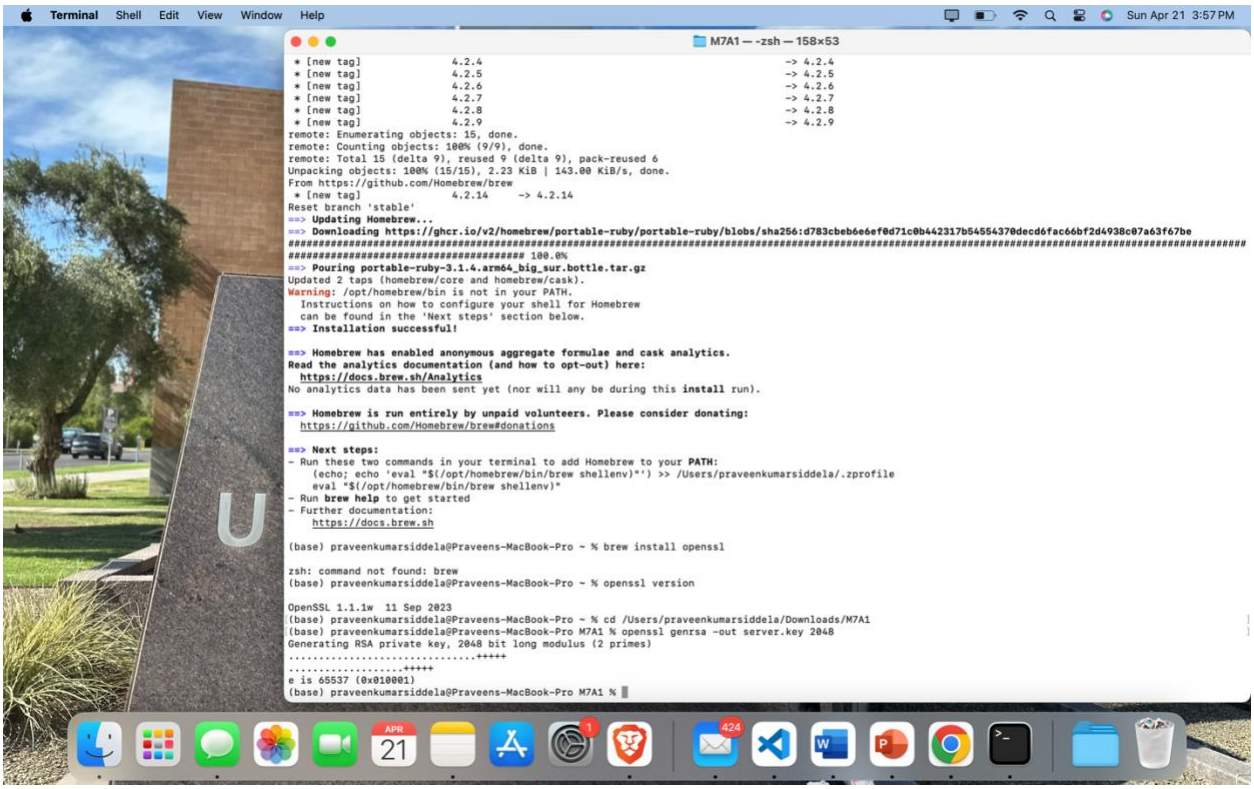


Screenshot 3: Showing execution of command to generate server.cer

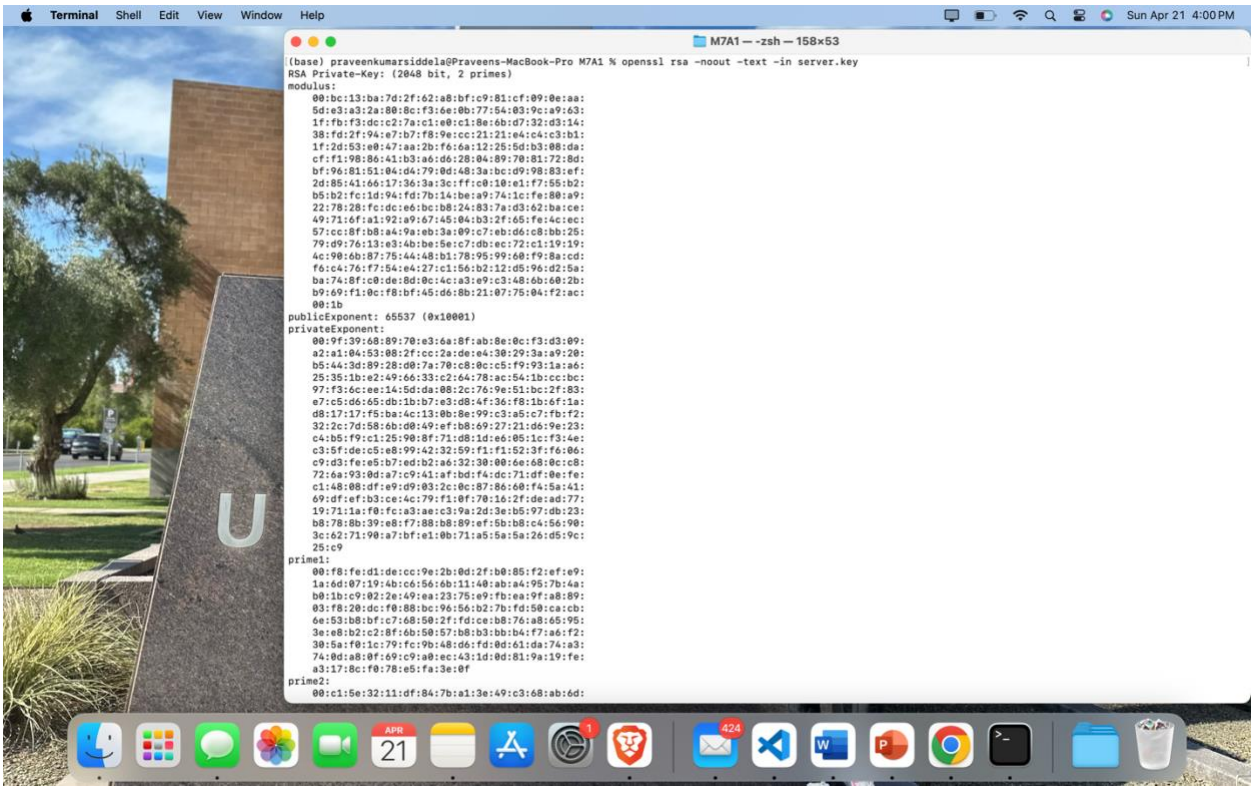


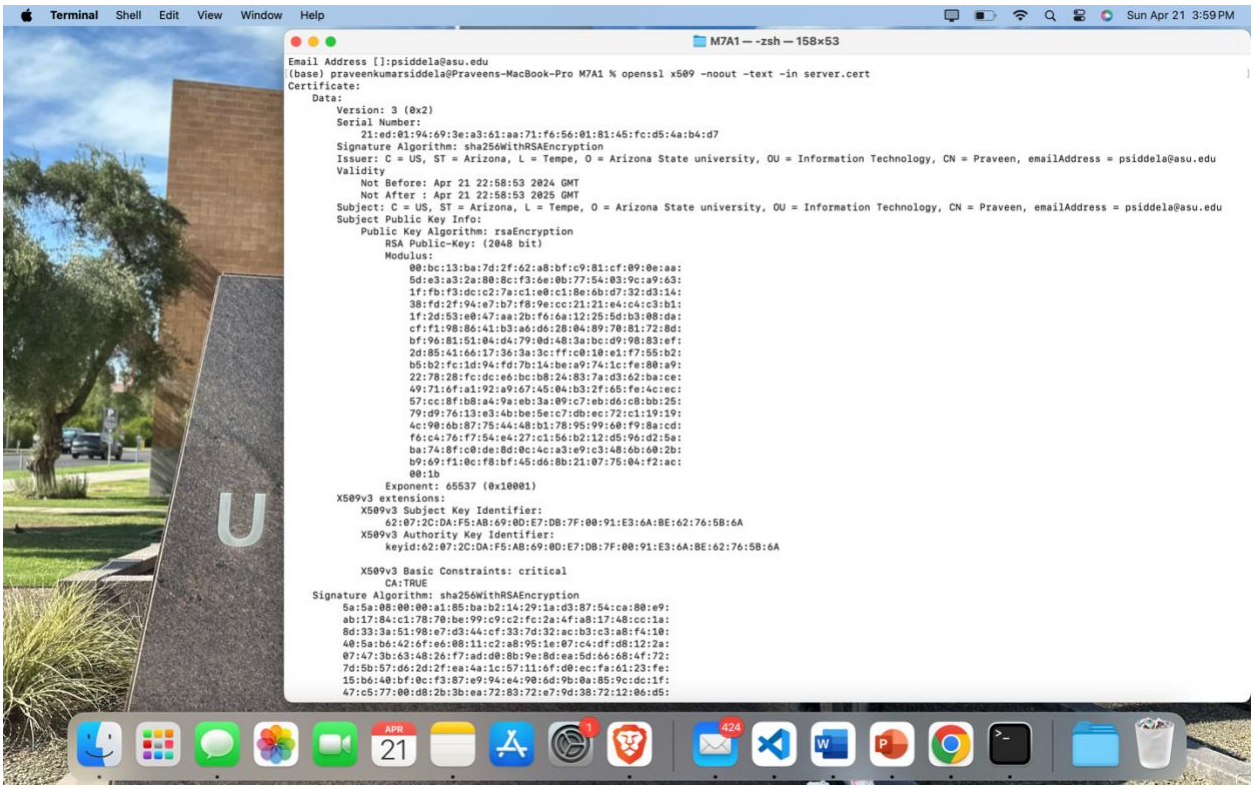
Screenshot 4: Showing filled-in certificate information prompts.



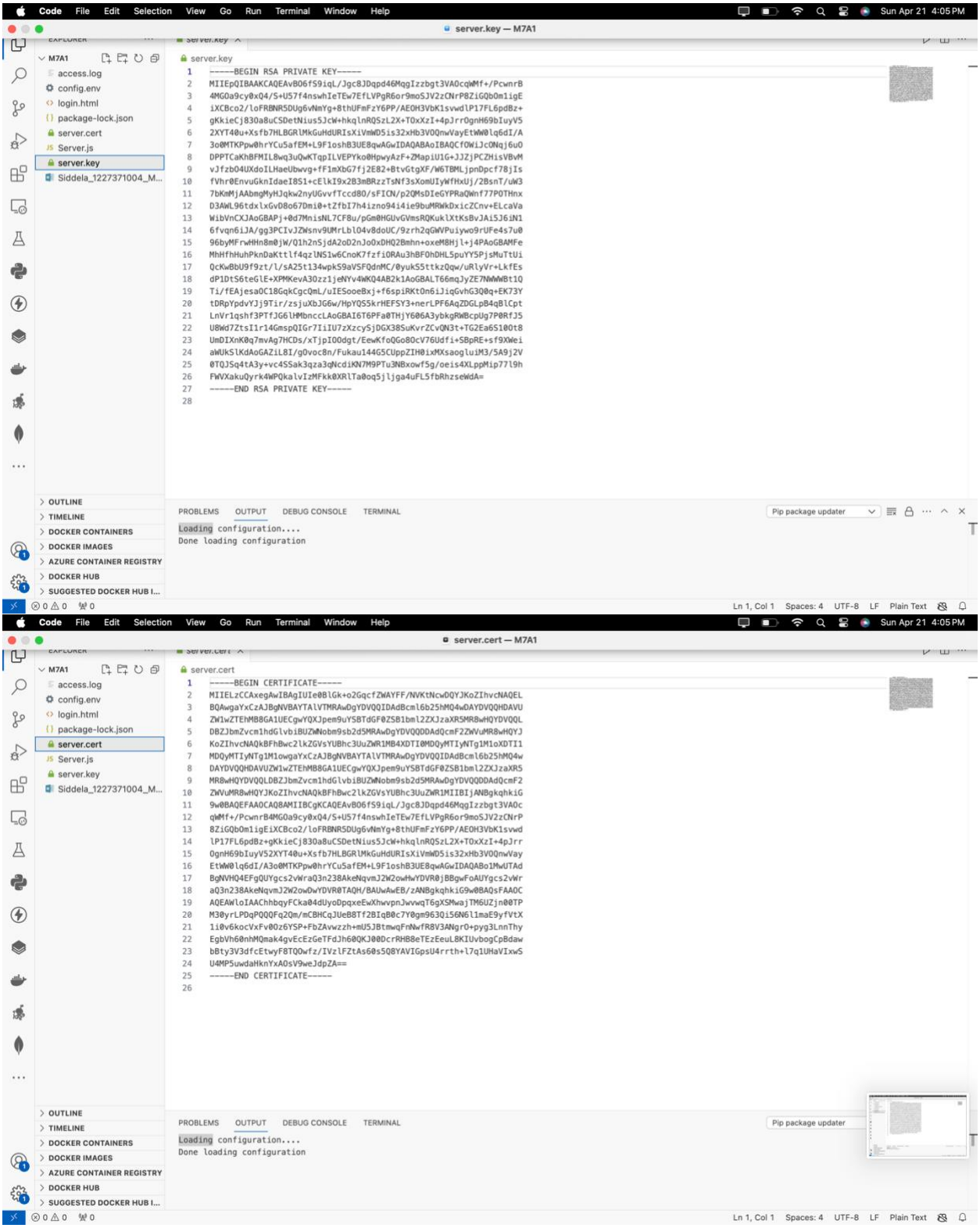


Screenshot 5: Showing output of verification commands in Command Prompt.

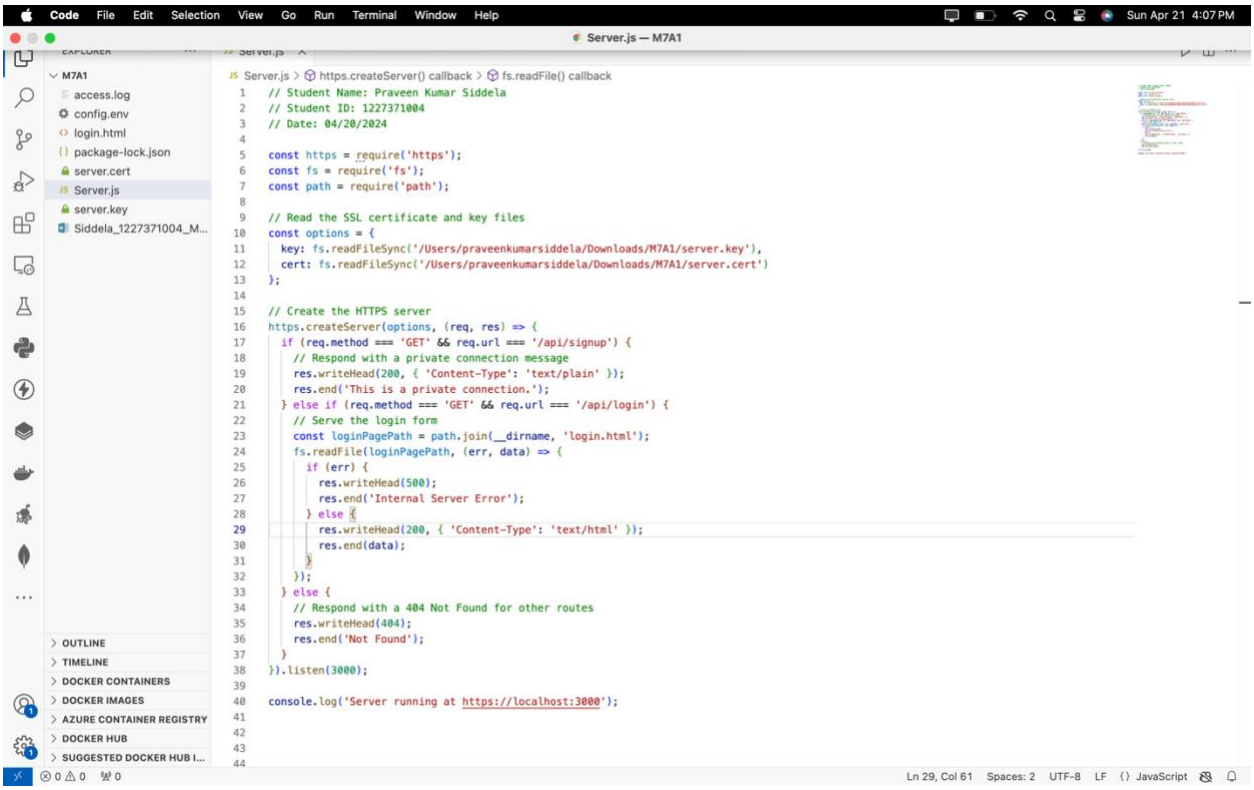



Terminal Shell Edit View Window Help
Sun Apr 21 3:59 PM
M7A1 — -zsh — 158×53
Email Address []:psiddela@asu.edu
(base) praveenkumarsiddela@Praveens-MacBook-Pro M7A1 % openssl x509 -noout -text -in server.cert
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
21:ed:01:94:69:3e:a3:61:aa:71:f6:56:01:81:45:fc:d5:4a:b4:d7
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = US, ST = Arizona, L = Tempe, O = Arizona State university, OU = Information Technology, CN = Praveen, emailAddress = psiddela@asu.edu
Validity
Not Before: Apr 21 22:58:53 2024 GMT
Not After : Apr 21 22:58:53 2025 GMT
Subject: C = US, ST = Arizona, L = Tempe, O = Arizona State university, OU = Information Technology, CN = Praveen, emailAddress = psiddela@asu.edu
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public-Key: (2048 bit)
Modulus:
00:bc:13:ba:7d:2f:62:a8:bf:c9:81:cf:09:0e:aa:
5d:e3:a3:2a:80:8c:f3:6e:0b:77:54:03:9c:a9:63:
1f:fb:f3:dc:c2:7a:c1:e0:c1:8e:6b:d7:32:d3:14:
38:fd:2f:94:e7:b7:f8:9e:cc:21:21:e4:c4:c3:b1:
1f:2d:53:e0:47:aa:2b:f6:6a:12:25:5d:b3:08:da:
cf:f1:98:86:41:b3:a6:d6:28:04:89:70:81:72:8d:
bf:96:81:51:04:d4:79:0d:48:3a:bc:d9:98:83:ef:
2d:85:41:66:17:36:3a:3c:ff:c0:10:e1:f7:55:b2:
b5:b2:fc:1d:94:fd:7b:14:be:a9:74:1c:fe:80:a9:
22:78:28:fc:dc:e6:bc:b8:24:83:7a:d3:62:ba:ce:
49:71:6f:a1:92:a9:67:45:04:b3:2f:65:fe:4c:ec:
57:cc:8f:b8:a4:9a:eb:3a:09:c7:eb:d6:c8:bb:25:
79:d9:76:13:e3:4b:be:5e:c7:db:ec:72:c1:19:19:
4c:90:6b:87:75:44:48:b1:78:95:99:60:f9:8a:cd:
f6:c4:76:f7:54:e4:27:c1:56:b2:12:d5:96:d2:5a:
ba:74:8f:c0:de:8d:0c:4c:a3:e9:c3:48:6b:60:2b:
b9:69:f1:0c:f8:bf:45:d6:8b:21:07:75:04:f2:ac:
00:1b
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Subject Key Identifier:
62:07:2C:DA:F5:AB:69:0D:E7:DB:7F:00:91:E3:6A:BE:62:76:5B:6A
X509v3 Authority Key Identifier:
keyid:62:07:2C:DA:F5:AB:69:0D:E7:DB:7F:00:91:E3:6A:BE:62:76:5B:6A
X509v3 Basic Constraints: critical
CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
5a:5a:08:00:00:a1:85:ba:b2:14:29:1a:d3:87:54:ca:80:e9:
ab:17:84:c1:78:70:be:99:c9:c2:fc:2a:4f:a8:17:48:cc:1a:
8d:33:3a:51:98:e7:d3:44:cf:33:7d:32:ac:b3:c3:a8:f4:10:
40:5a:b6:42:6f:e6:08:11:c2:a8:95:1e:07:c4:df:d8:12:2a:
07:47:3b:63:48:26:f7:ad:d0:8b:9e:8d:ea:5d:66:68:4f:72:
7d:5b:57:d6:2d:2f:ea:4a:1c:57:11:6f:d0:ec:fa:61:23:fe:
15:b6:40:bf:0c:f3:87:e9:94:e4:90:6d:9b:0a:85:9c:dc:1f:
47:c6:77:00:d8:2b:3b:ea:72:83:72:e7:9d:38:72:12:06:d5:

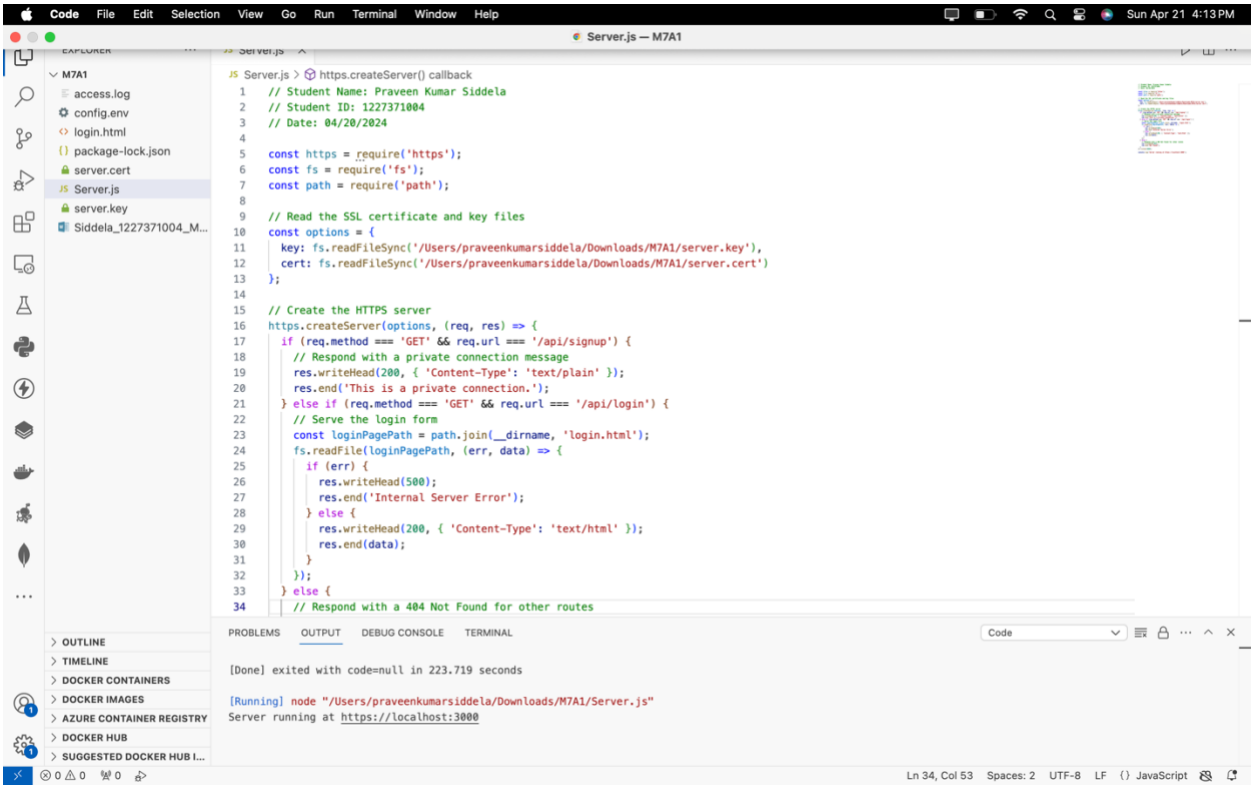
Screenshot 6: Of the project folder with `server.key` and `server.cert` files.



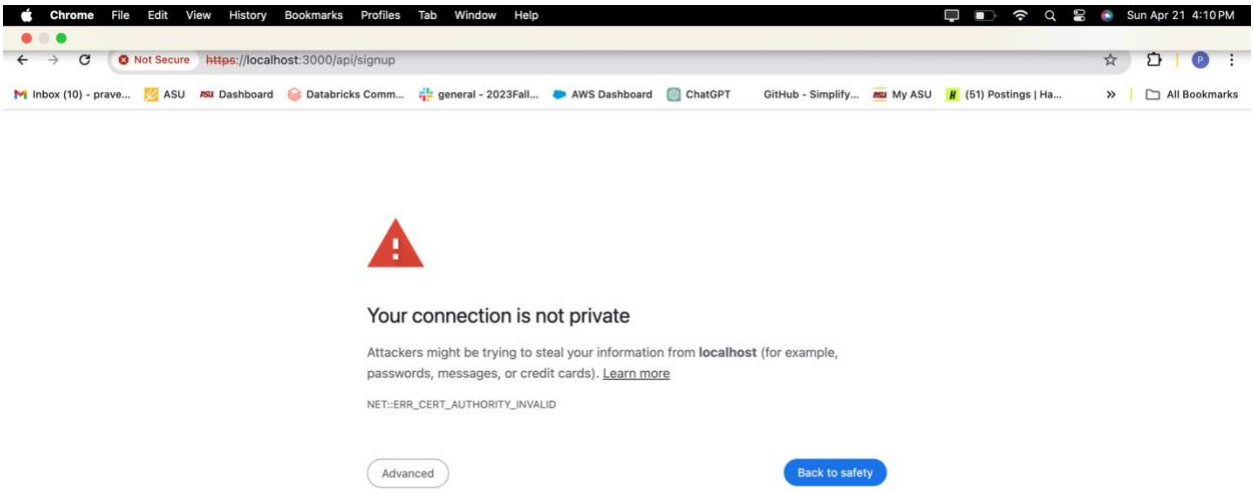
Screenshot 7: Showing the modified Server.js file in your IDE.



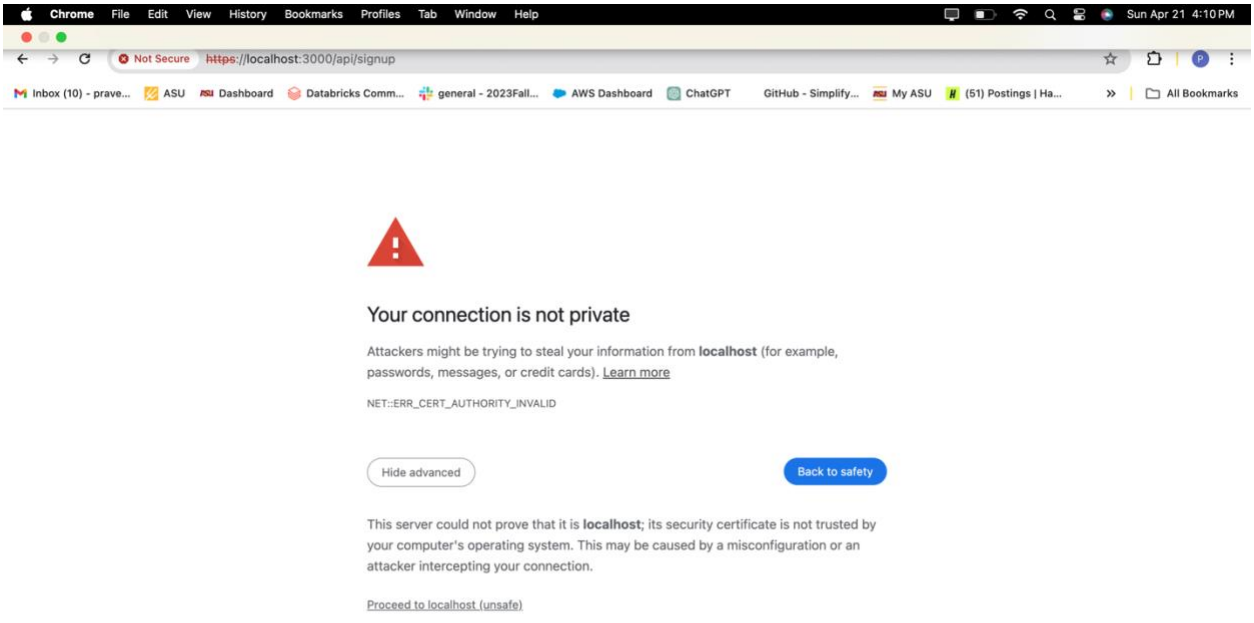
Screenshot 8: Showing the terminal with the running Node.js server.



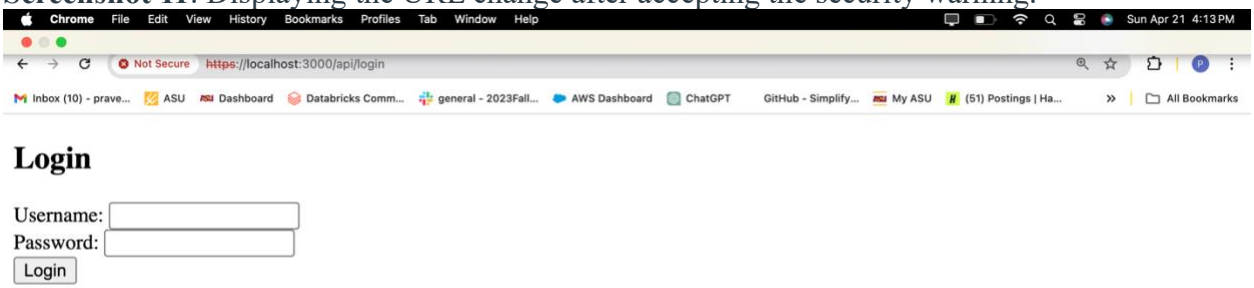
Screenshot 9: Showing the browser view when accessing the HTTPS server.

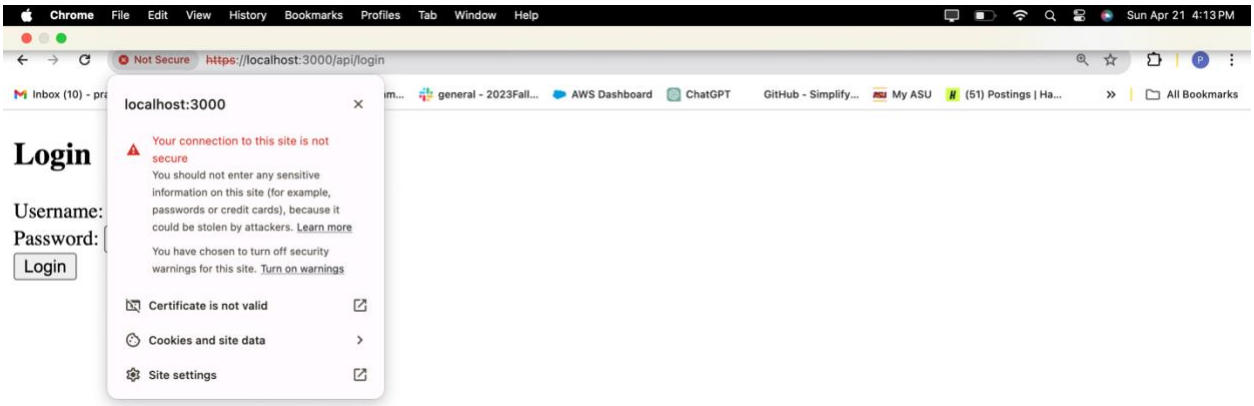


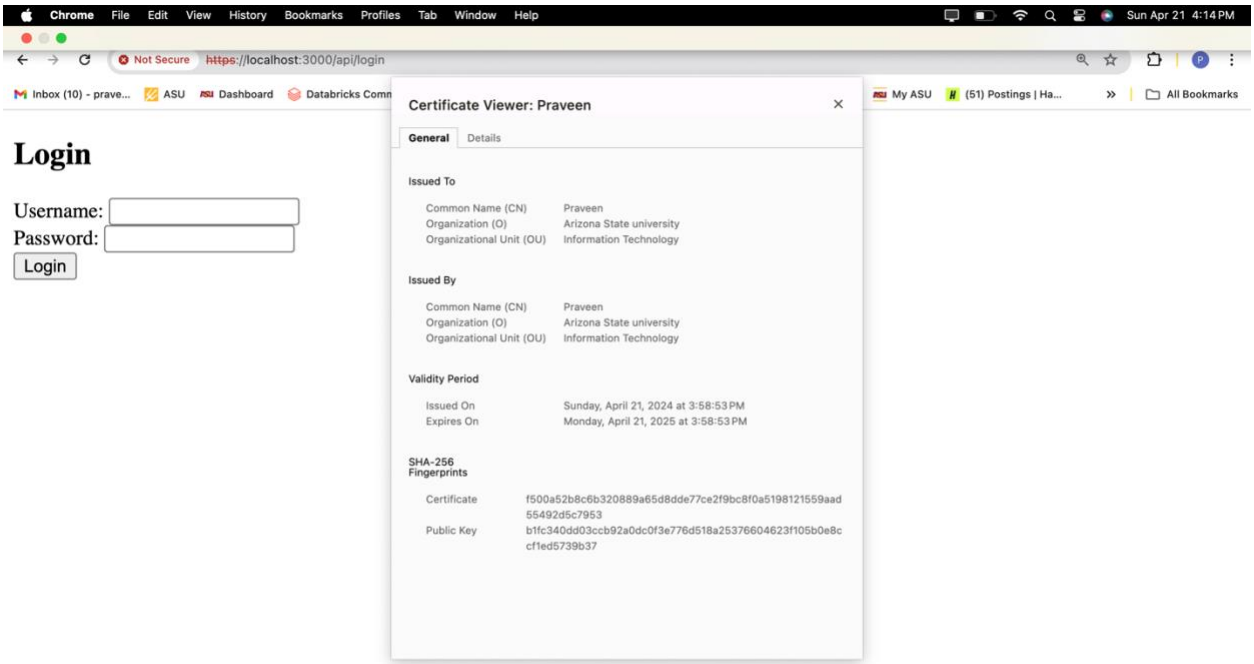
Screenshot 10: Showing the browser's security warning.



Screenshot 11: Displaying the URL change after accepting the security warning.

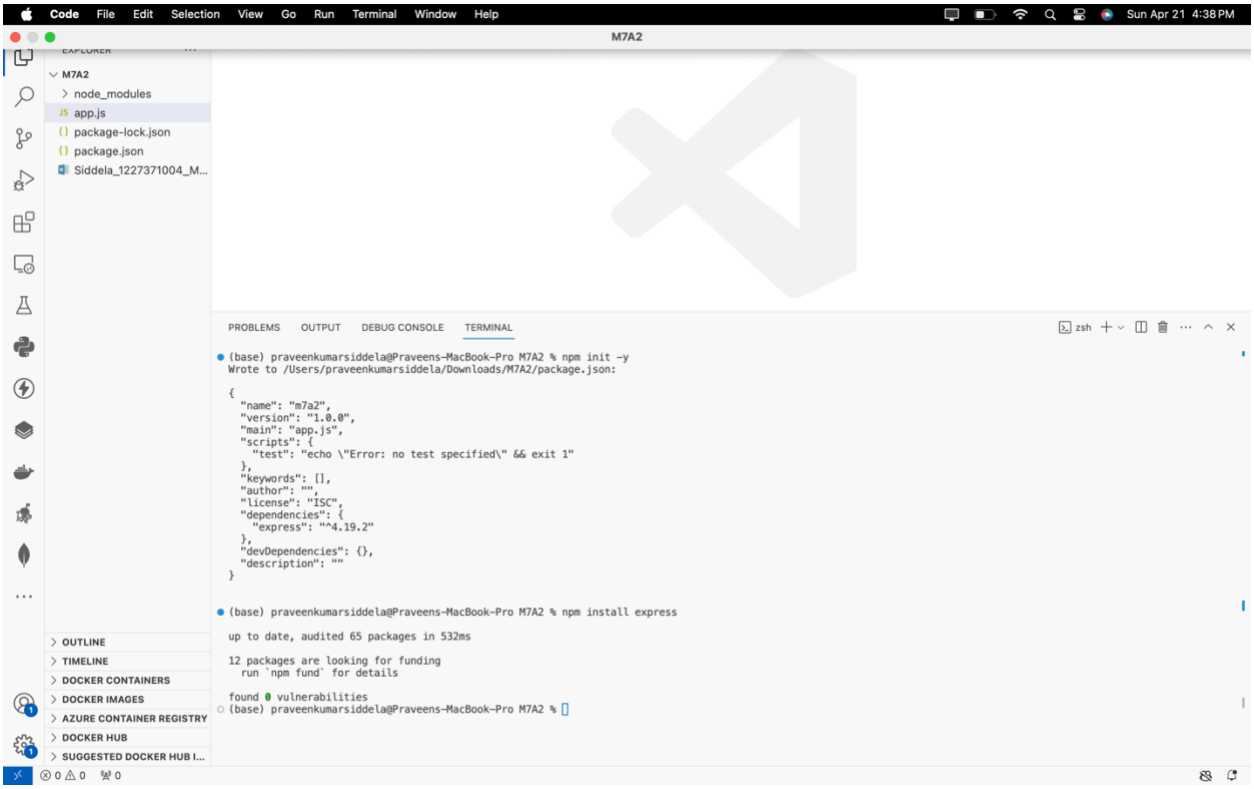




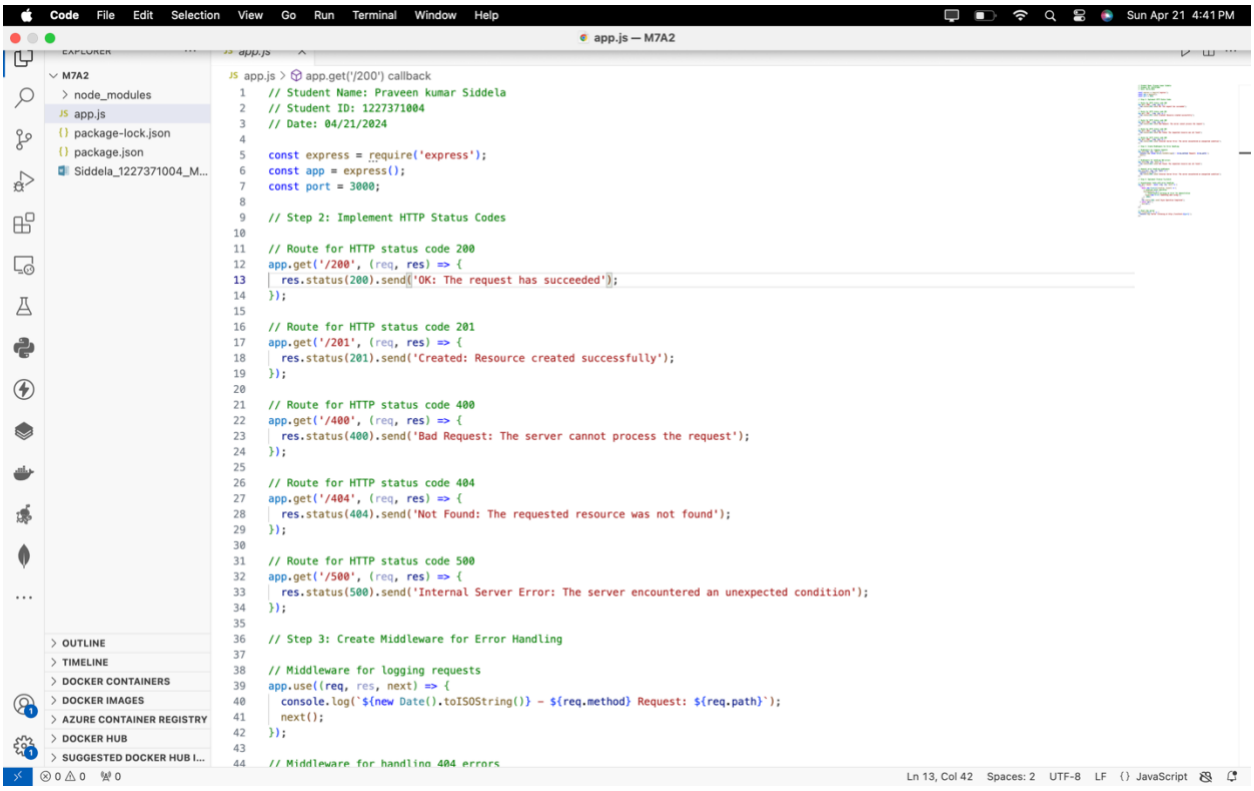

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 4:14 PM
Not Secure https://localhost:3000/api/login
Inbox (10) - prave... ASU Dashboard Databricks Comm
My ASU (51) Postings | Ha... All Bookmarks
Login
Username:
Password:
Login
Certificate Viewer: Praveen
General Details
Issued To
Common Name (CN) Praveen
Organization (O) Arizona State university
Organizational Unit (OU) Information Technology
Issued By
Common Name (CN) Praveen
Organization (O) Arizona State university
Organizational Unit (OU) Information Technology
Validity Period
Issued On Sunday, April 21, 2024 at 3:58:53 PM
Expires On Monday, April 21, 2025 at 3:58:53 PM
SHA-256 Fingerprints
Certificate f500a52b8c6b320889a65d8dde77ce2f9bc8f0a5198121559aad55492d5c7953
Public Key b1fc340dd03ccb92a0dc0f3e776d518a25376604623f105b0e8ccf1ed5739b37

Module 7 Activity 2: Implementing HTTP Status Codes and Error Handling in Node.js

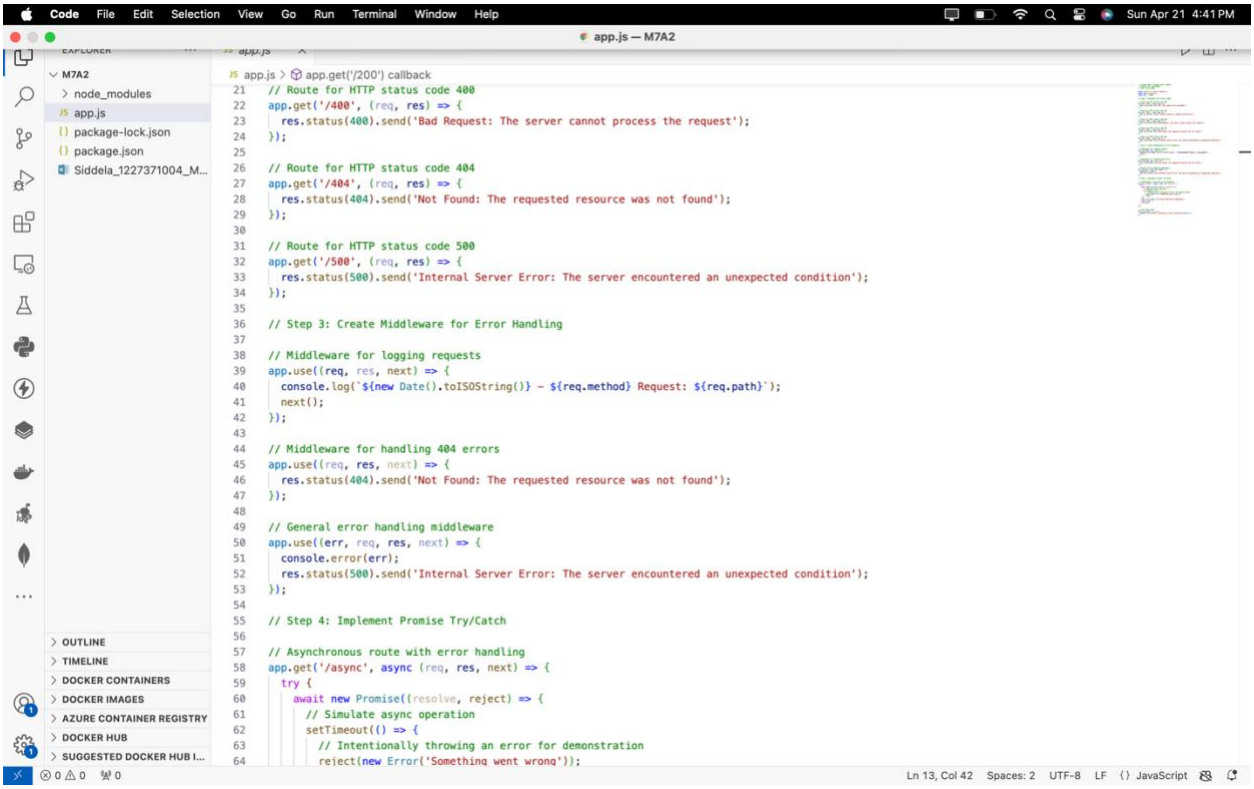
Screenshot 1: Of the initialized project structure in your IDE.



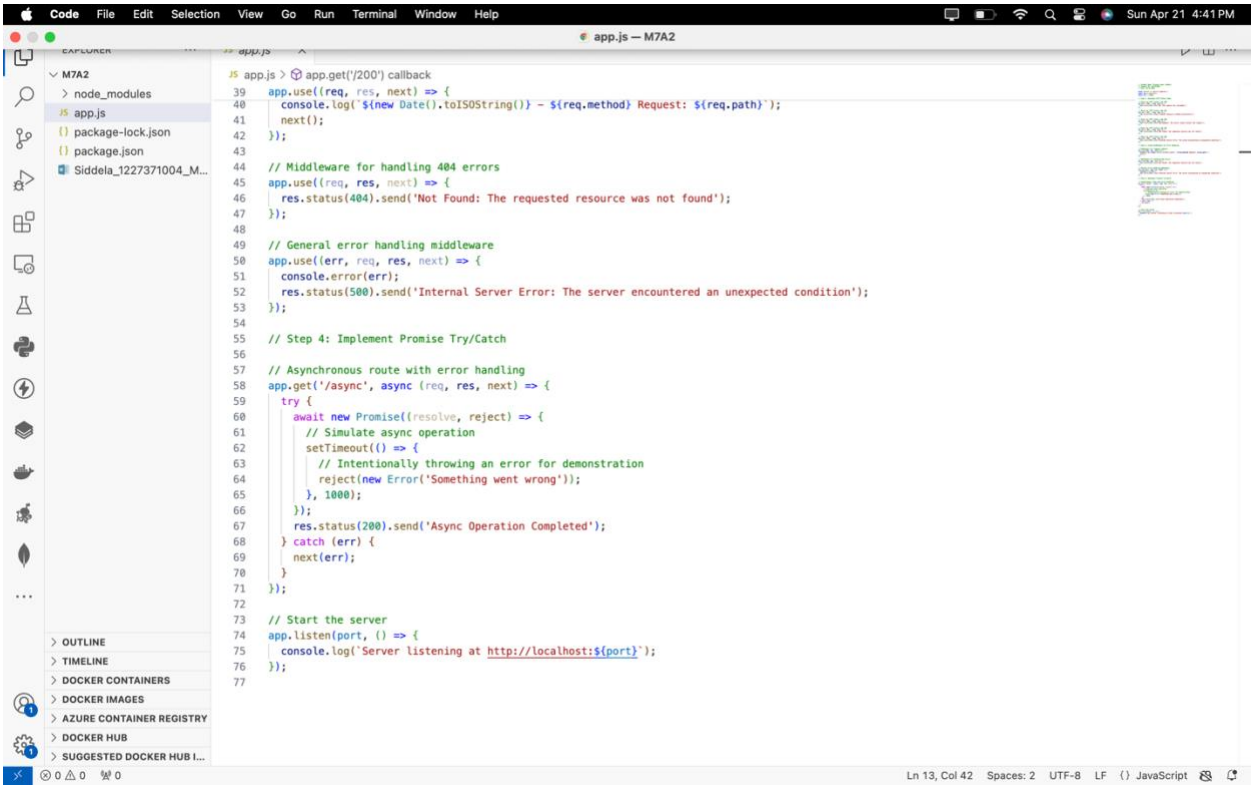
Screenshot 2: Showing the app.js file implemented routes for different HTTP status codes.



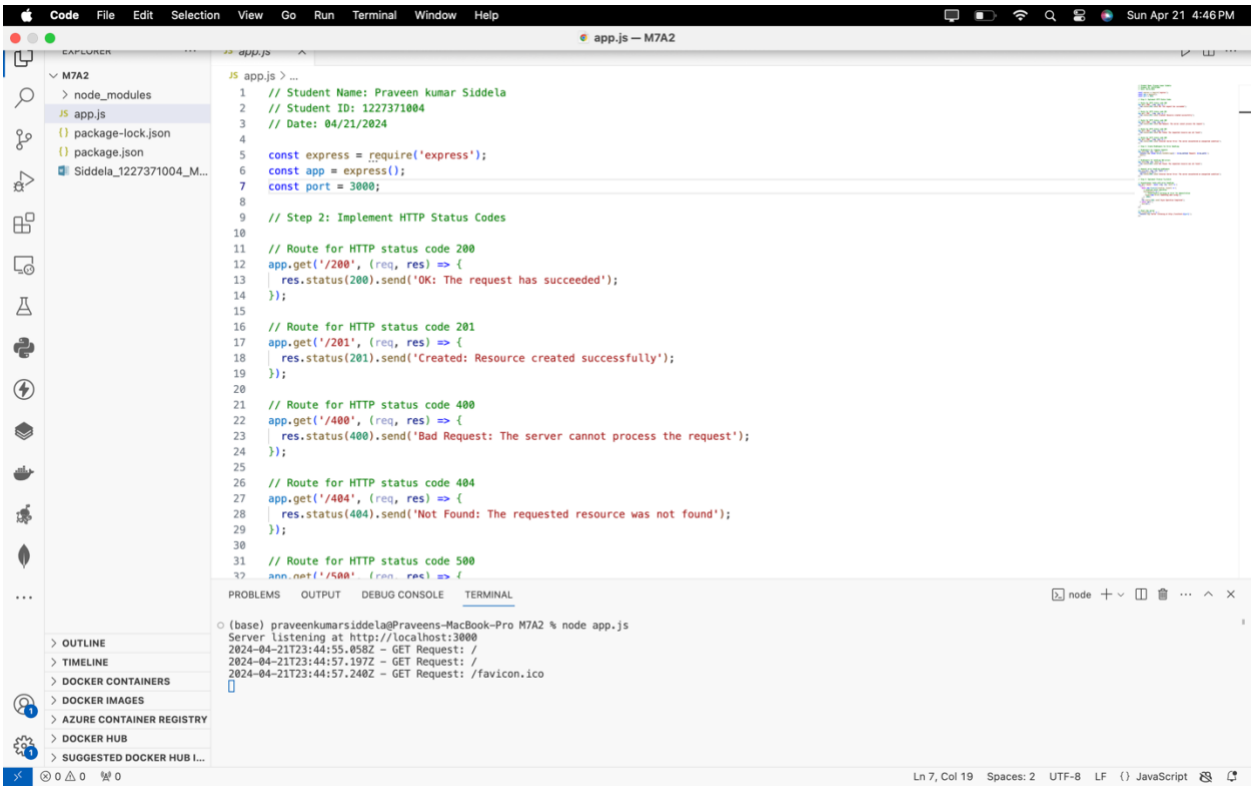
Screenshot 3: Displaying the implemented middleware in app.js for request logging and error handling.



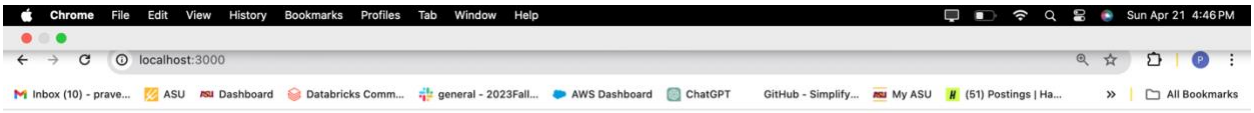
Screenshot 4: Of the asynchronous route with try/catch in app.js.



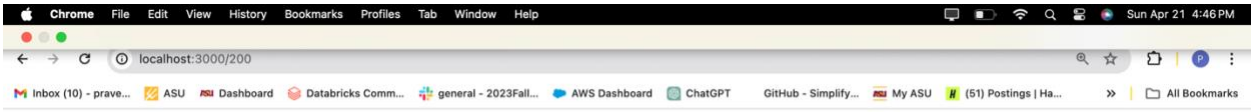
Screenshot 5: Showing the running server in the terminal.



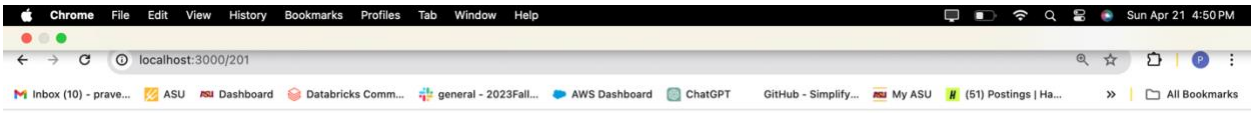
Screenshot 6: Displaying responses from the server for different routes in Postman or a web browser



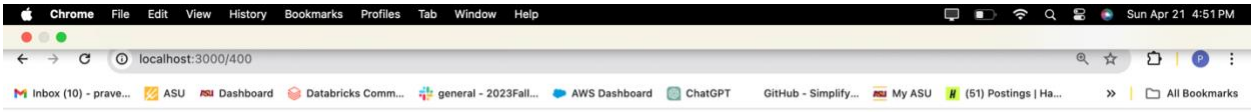
Not Found: The requested resource was not found



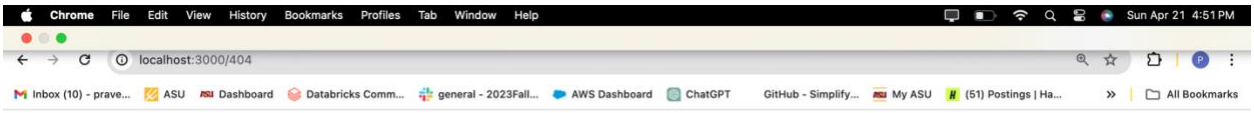
OK: The request has succeeded



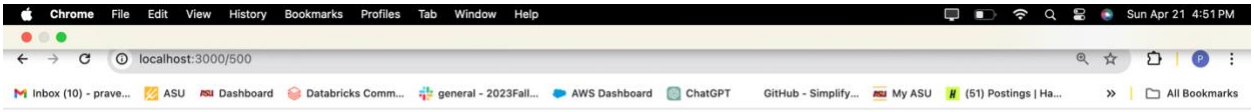
Created: Resource created successfully



Bad Request: The server cannot process the request



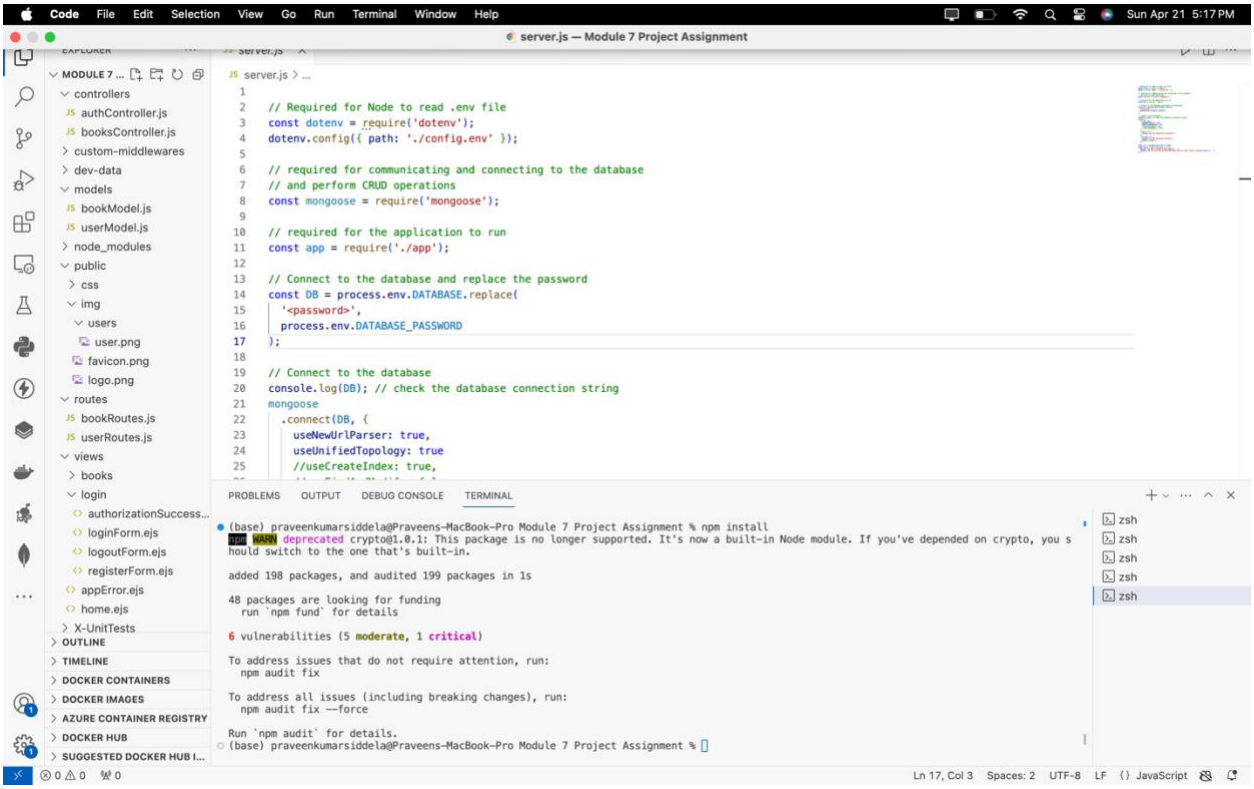
Not Found: The requested resource was not found



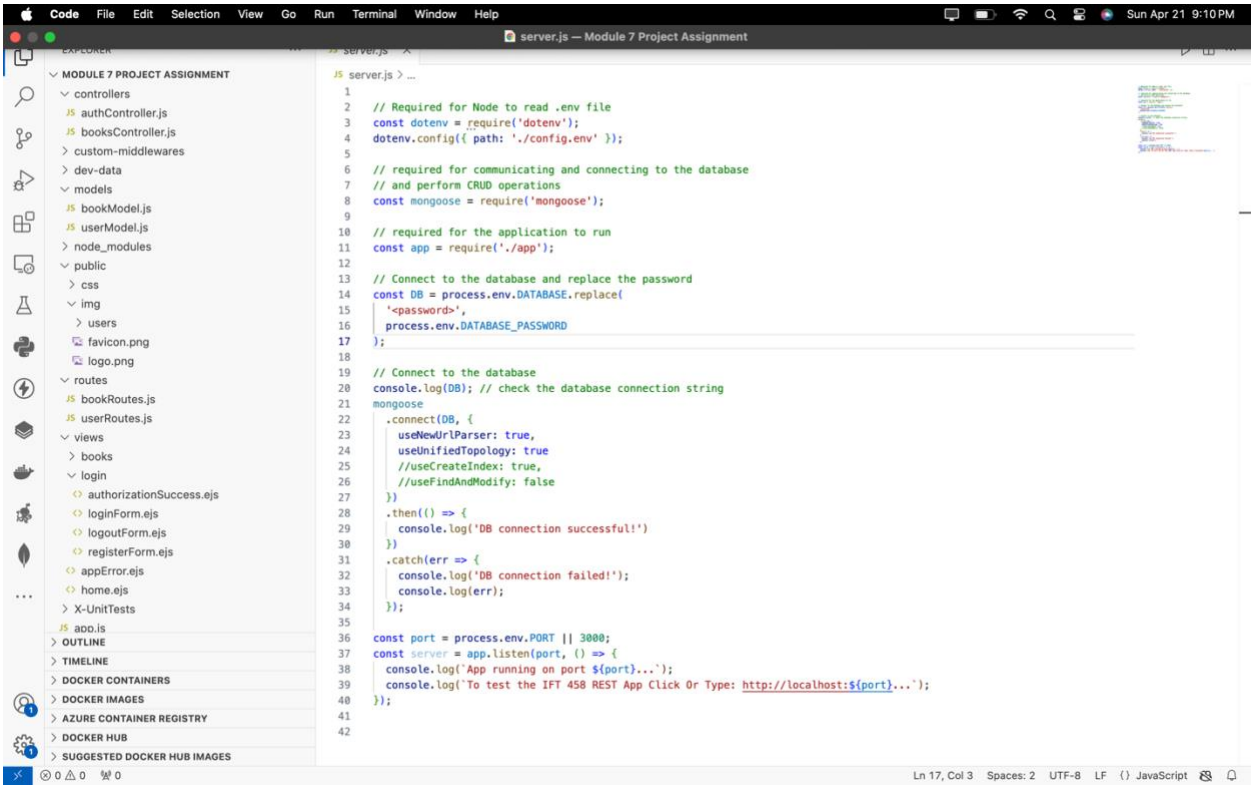
Internal Server Error: The server encountered an unexpected condition

Module 7 Lab 1: Develop Secure Middleware & Backend Database for a Book Exchange Application

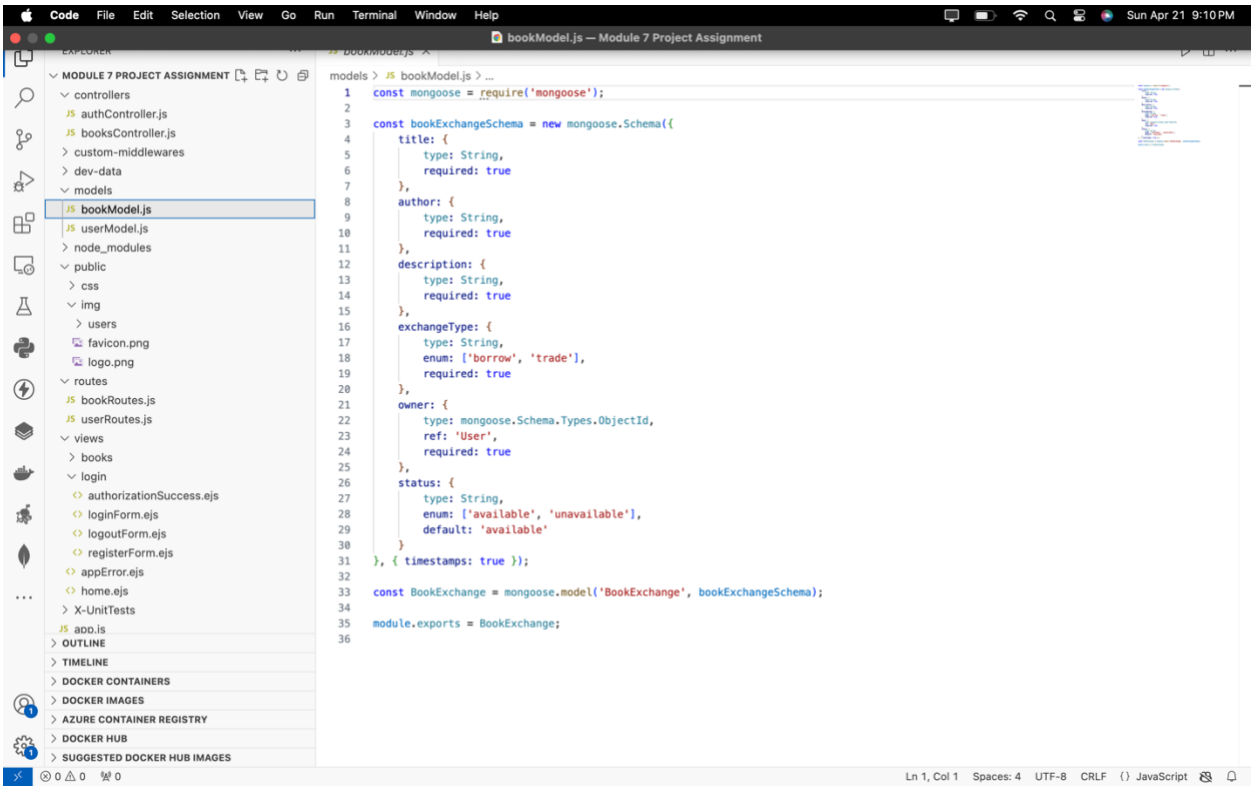
Screenshot 1: Of the terminal showing successful installation of dependencies.



Screenshot 2: Of the project directory structure showcasing MVC organization.



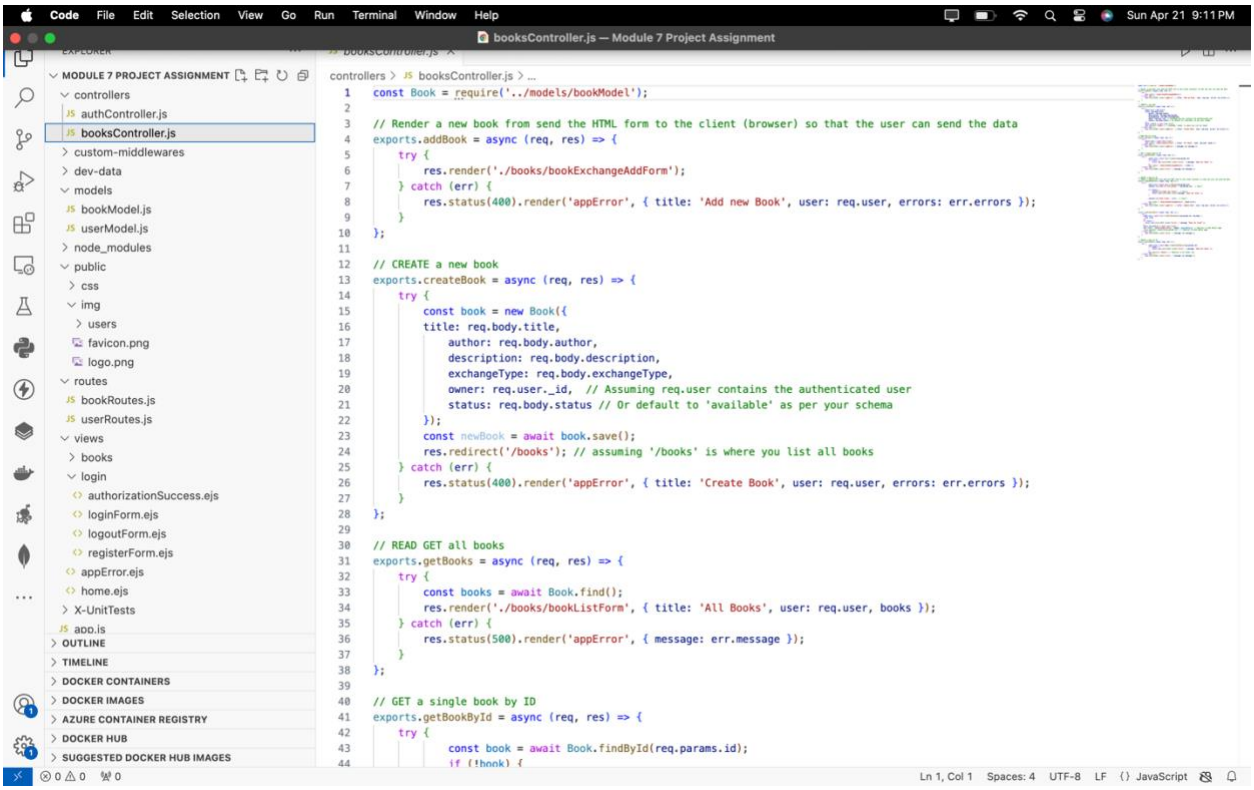
Screenshot 3: Of the BookModel.js file with a brief explanation.

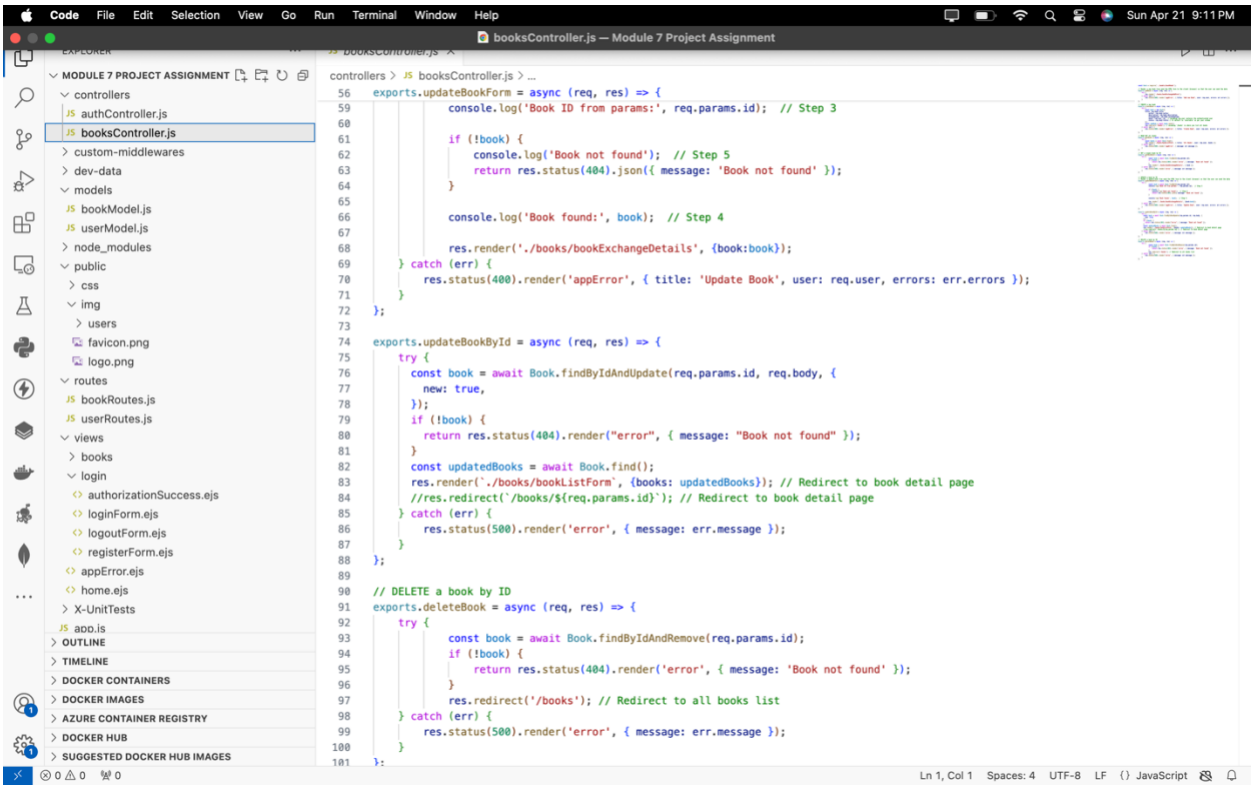


The **bookModel.js** file defines the book data structure for the application using Mongoose. It specifies properties like title, author, description, exchangeType, owner, and status, with precise data types and constraints. Enums ensure consistency for exchangeType and status. The owner field references the 'User' model. Default values and timestamps are included. The schema becomes a Mongoose model named 'BookExchange' for easy MongoDB interaction, exported for application-wide use.

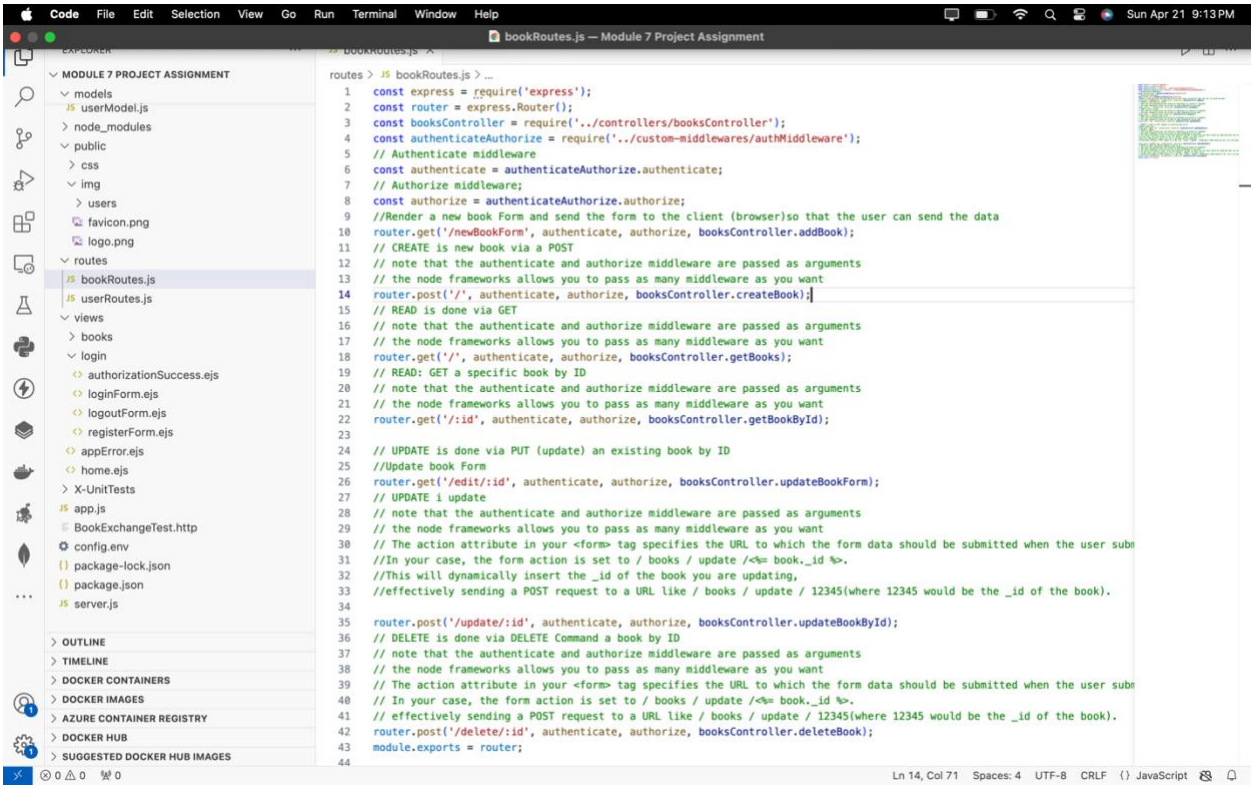
Screenshot 4: Of the bookController.js with the implemented logic.

The **bookController.js** file handles book-related actions in the app. It includes functions for adding, retrieving, updating, and deleting books. Error handling ensures appropriate status codes and messages. Functions like addBook and updateBookForm handle user input, while createBook, getBooks, getBookById, updateBookById, and deleteBook interact with the database. Status codes like 400, 404, and 500 are used for various scenarios, ensuring smooth operation of book-related functionalities.



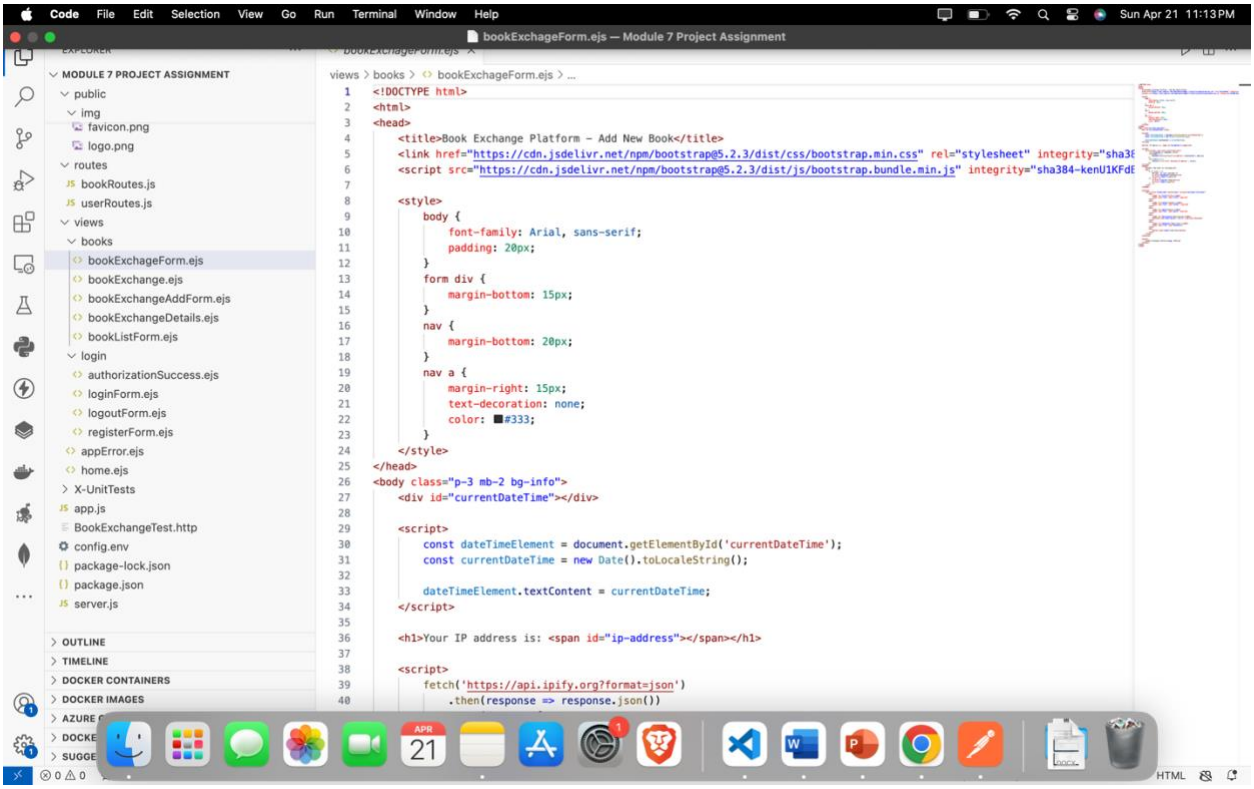


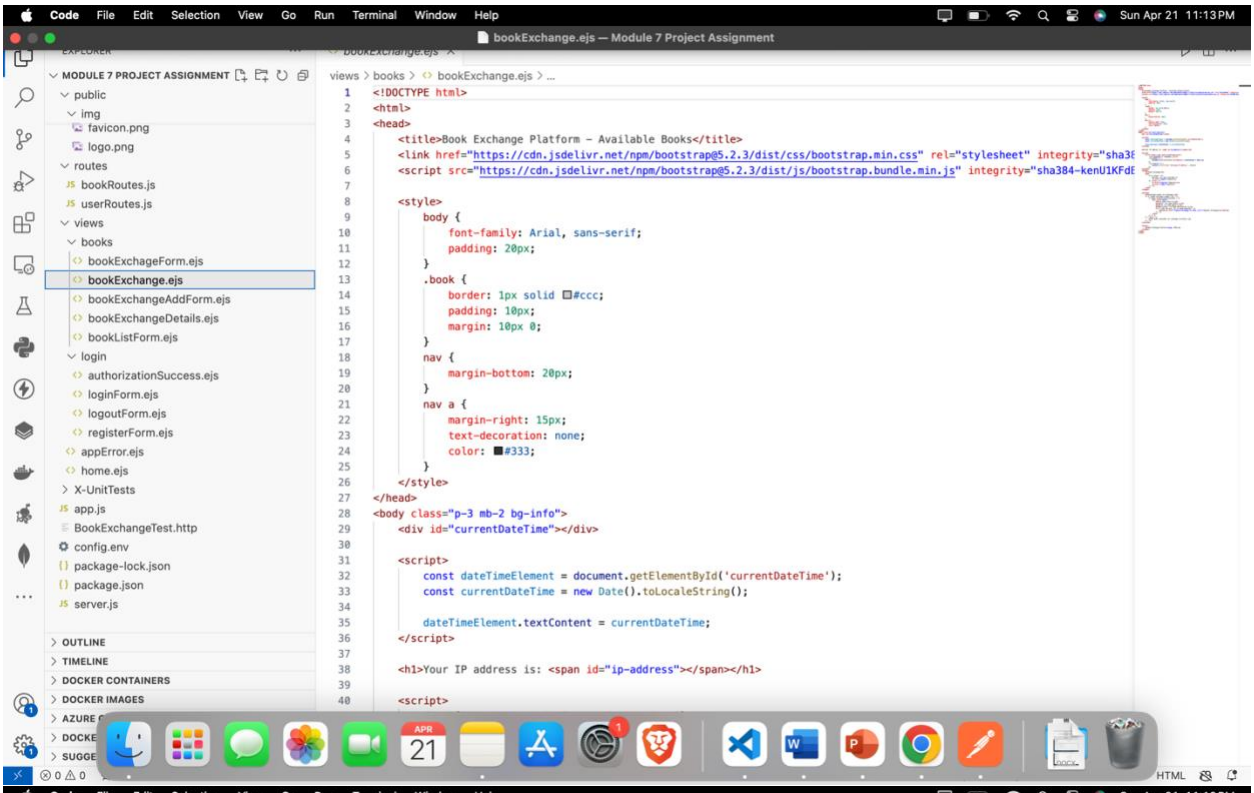
Screenshot 5: Of the bookRoutes.js file with defined routes.



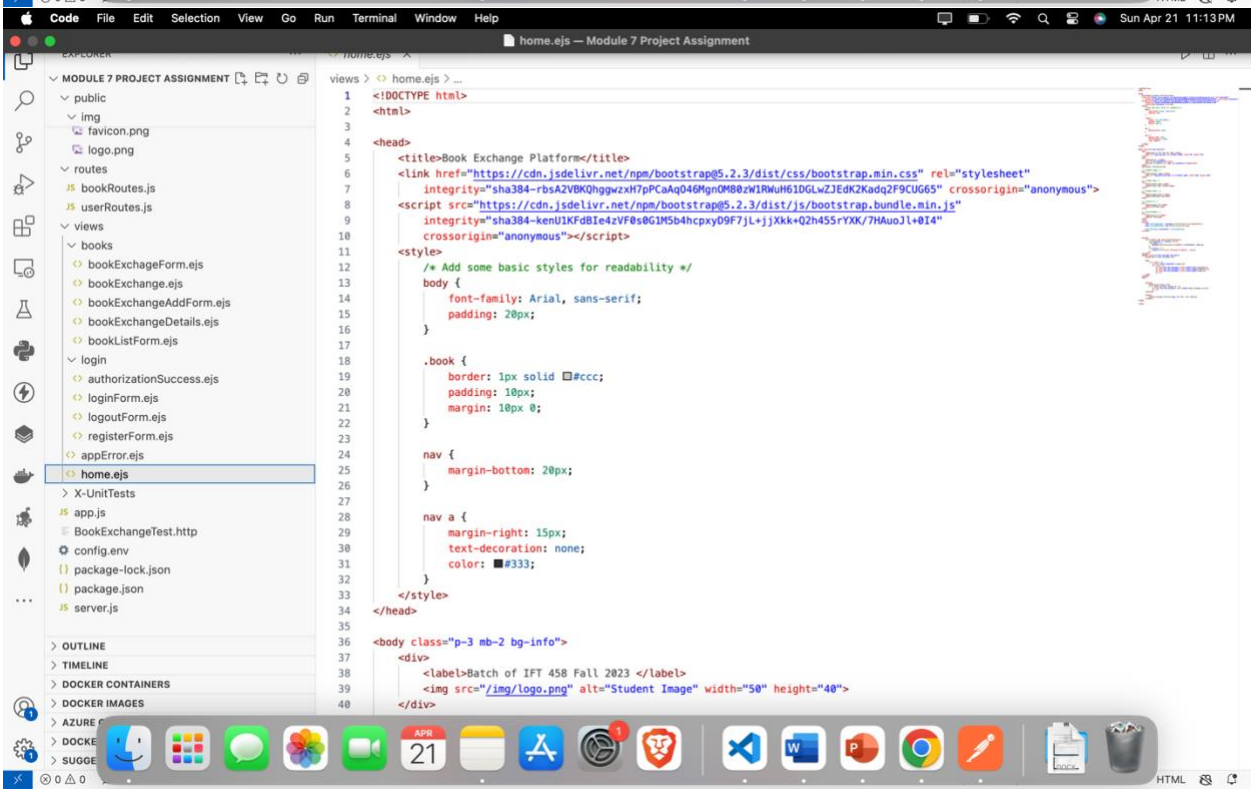
Screenshot 6: Of updated view files (*.ejs) and changes made.

The **bookExchangeForm.ejs** page is a web form for adding new books to a book exchange platform. It contains standard HTML structure with metadata and external resources like Bootstrap CSS and JavaScript files. The body section dynamically displays date, time, and user IP. When logged in, it shows a welcome message and links to dashboard and logout; otherwise, it provides signup/login buttons. The main section houses a form for entering book details and submitting them to **/books/add**. The footer includes copyright information. Overall, it serves as a user-friendly interface for adding books to the platform.




Code File Edit Selection View Go Run Terminal Window Help
Sun Apr 21 11:13 PM
bookExchange.ejs — Module 7 Project Assignment
MODULE 7 PROJECT ASSIGNMENT
public
img
favicon.png
logo.png
routes
bookRoutes.js
userRoutes.js
views
books
bookExchageForm.ejs
bookExchange.ejs
bookExchangeAddForm.ejs
bookExchangeDetails.ejs
bookListForm.ejs
login
authorizationSuccess.ejs
loginForm.ejs
logoutForm.ejs
registerForm.ejs
appError.ejs
home.ejs
X-UnitTests
app.js
BookExchangeTest.http
config.env
package-lock.json
package.json
server.js
OUTLINE
TIMELINE
DOCKER CONTAINERS
DOCKER IMAGES
views > books > bookExchange.ejs > ...
<!DOCTYPE html>
<html>
<head>
<title>Book Exchange Platform - Available Books</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha3
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kenU1KFdE
<style>
body {
font-family: Arial, sans-serif;
padding: 20px;
}
.book {
border: 1px solid #ccc;
padding: 10px;
margin: 10px 0;
}
nav {
margin-bottom: 20px;
}
nav a {
margin-right: 15px;
text-decoration: none;
color: #333;
}
</style>
</head>
<body class="p-3 mb-2 bg-info">

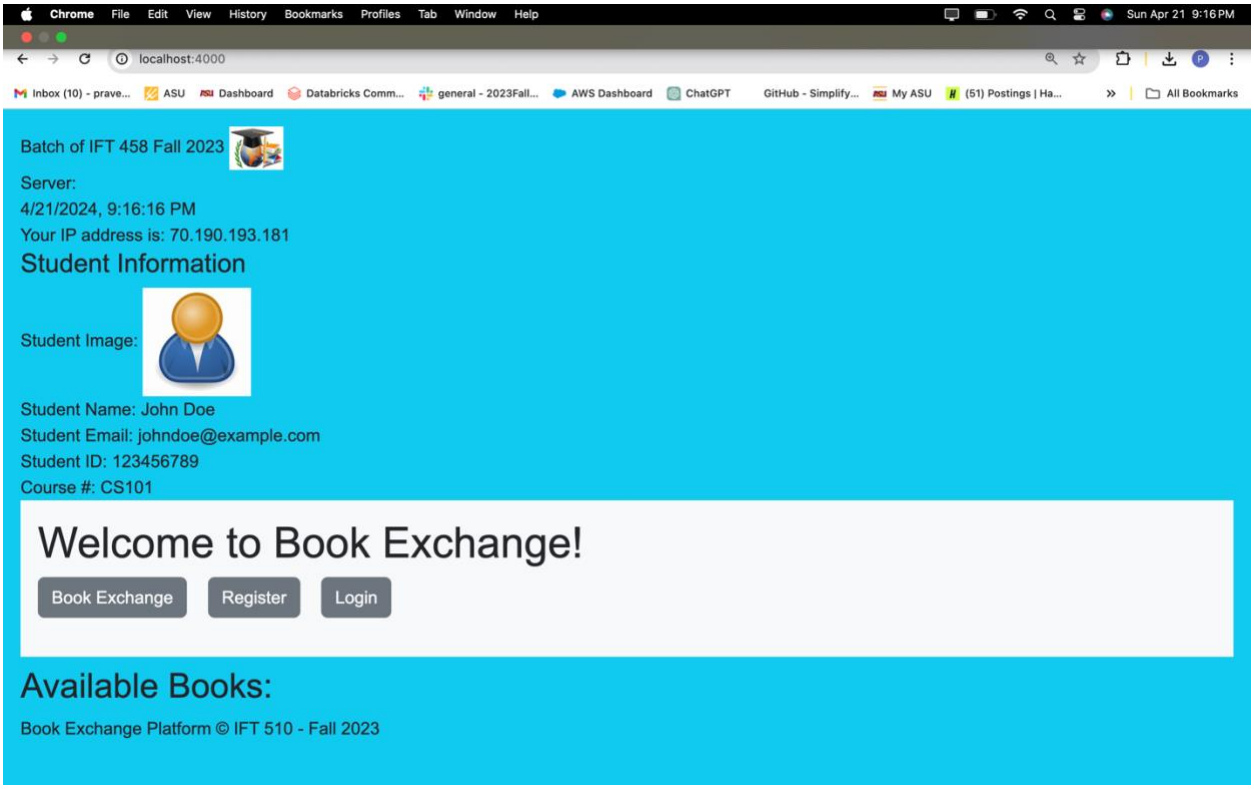
<script>
const dateTimeElement = document.getElementById('currentDateTime');
const currentDateTime = new Date().toLocaleString();
dateTimeElement.textContent = currentDateTime;
</script>
<h1>Your IP address is: </h1>
<script>
HTML


Code File Edit Selection View Go Run Terminal Window Help
Sun Apr 21 11:13 PM
home.ejs — Module 7 Project Assignment
MODULE 7 PROJECT ASSIGNMENT
public
img
favicon.png
logo.png
routes
bookRoutes.js
userRoutes.js
views
books
bookExchageForm.ejs
bookExchange.ejs
bookExchangeAddForm.ejs
bookExchangeDetails.ejs
bookListForm.ejs
login
authorizationSuccess.ejs
loginForm.ejs
logoutForm.ejs
registerForm.ejs
appError.ejs
home.ejs
X-UnitTests
app.js
BookExchangeTest.http
config.env
package-lock.json
package.json
server.js
OUTLINE
TIMELINE
DOCKER CONTAINERS
DOCKER IMAGES
views > home.ejs > ...
<!DOCTYPE html>
<html>
<head>
<title>Book Exchange Platform</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4"
crossorigin="anonymous"></script>
<style>
/* Add some basic styles for readability */
body {
font-family: Arial, sans-serif;
padding: 20px;
}
.book {
border: 1px solid #ccc;
padding: 10px;
margin: 10px 0;
}
nav {
margin-bottom: 20px;
}
nav a {
margin-right: 15px;
text-decoration: none;
color: #333;
}
</style>
</head>
<body class="p-3 mb-2 bg-info">

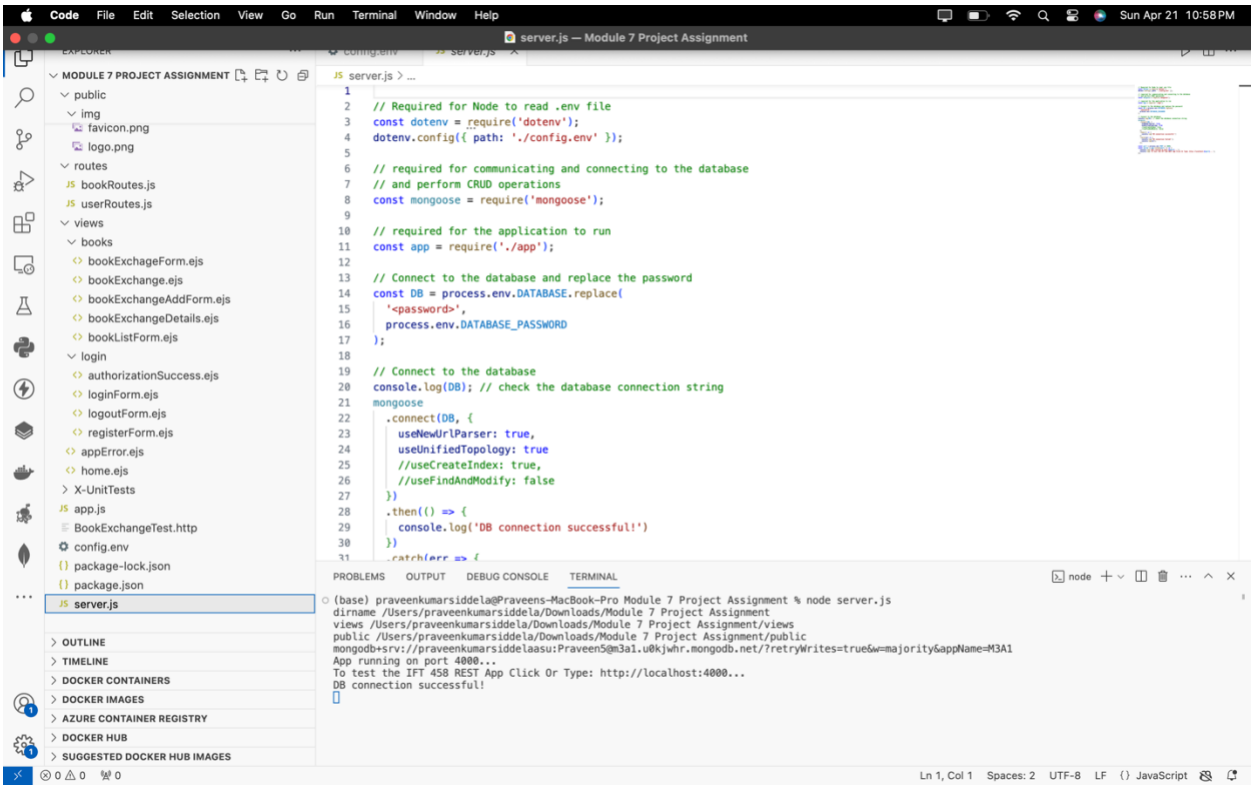
<label>Batch of IFT 458 Fall 2023 </label>
<img src="/img/logo.png" alt="Student Image" width="50" height="40">

HTML

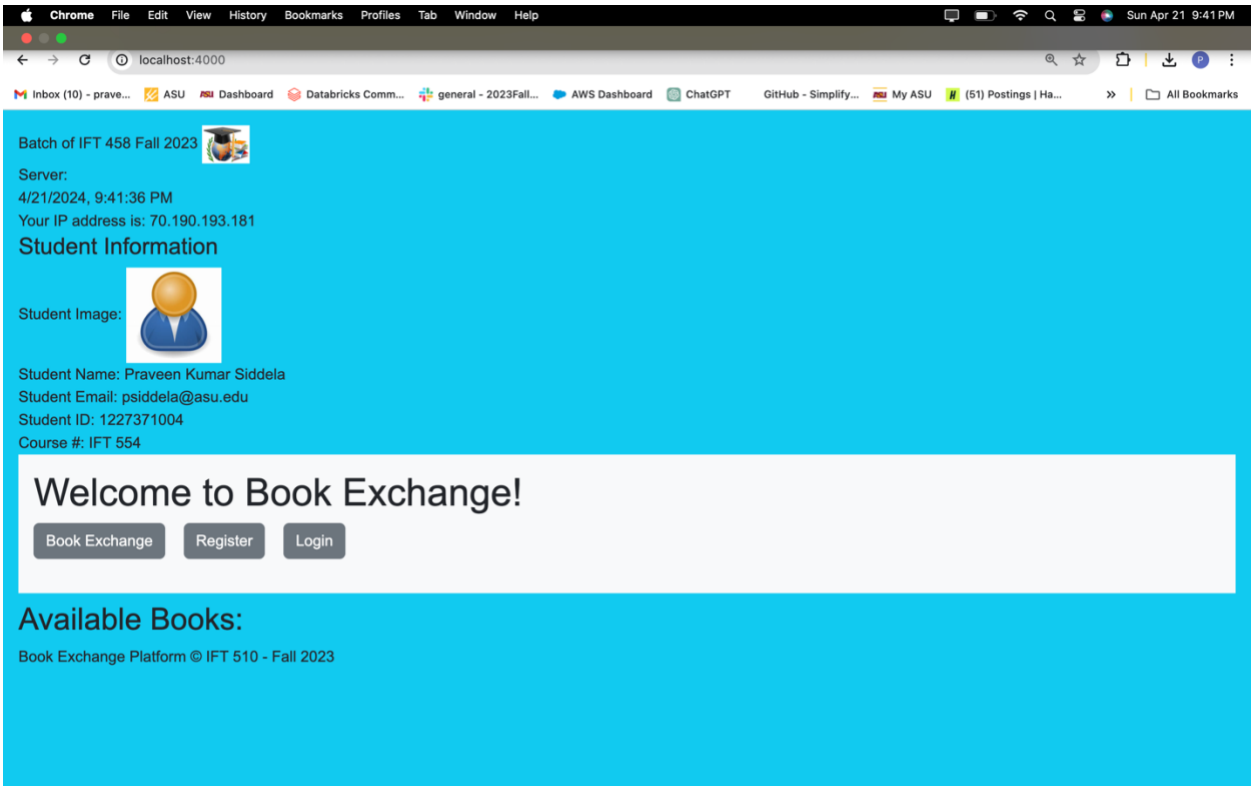
Screenshot 7: Of the home page as displayed in the browser.



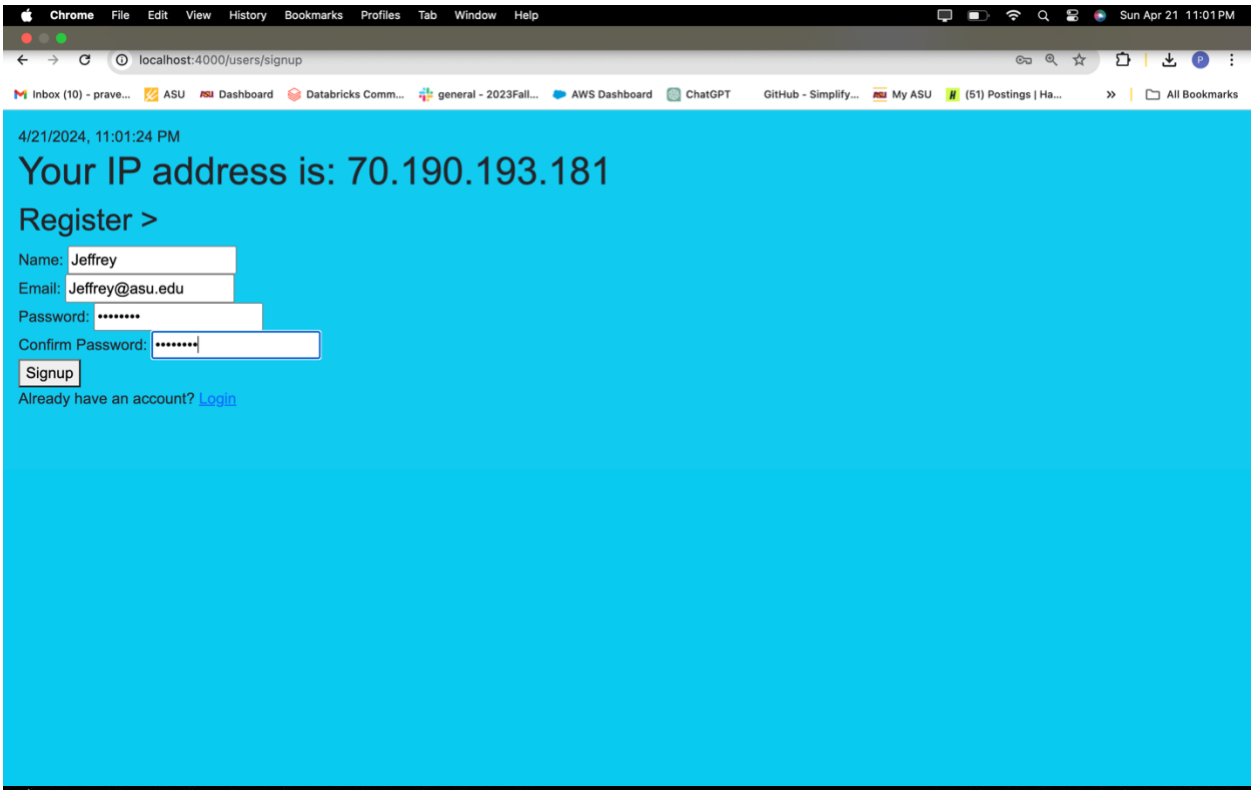
Screenshot 8: Of all personalized user interface pages.

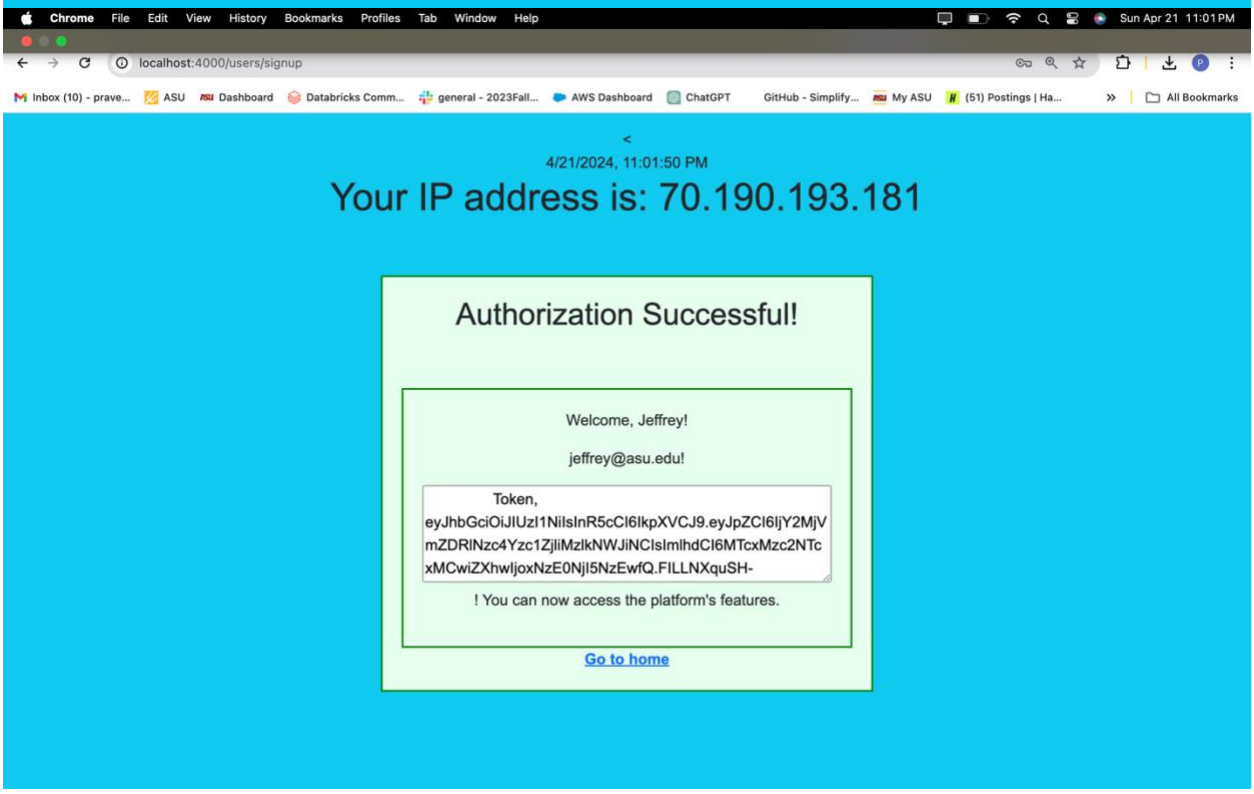

Code File Edit Selection View Go Run Terminal Window Help
Sun Apr 21 10:58 PM
server.js — Module 7 Project Assignment
MODULE 7 PROJECT ASSIGNMENT
public
img
favicon.png
logo.png
routes
bookRoutes.js
userRoutes.js
views
books
bookExchageForm.ejs
bookExchange.ejs
bookExchangeAddForm.ejs
bookExchangeDetails.ejs
bookListForm.ejs
login
authorizationSuccess.ejs
loginForm.ejs
logoutForm.ejs
registerForm.ejs
appError.ejs
home.ejs
X-UnitTests
app.js
BookExchangeTest.http
config.env
package-lock.json
package.json
server.js
// Required for Node to read .env file
const dotenv = require('dotenv');
dotenv.config({ path: './config.env' });
// required for communicating and connecting to the database
// and perform CRUD operations
const mongoose = require('mongoose');
// required for the application to run
const app = require('./app');
// Connect to the database and replace the password
const DB = process.env.DATABASE.replace(
'<password>',
process.env.DATABASE_PASSWORD
);
// Connect to the database
console.log(DB); // check the database connection string
mongoose
.connect(DB, {
useNewUrlParser: true,
useUnifiedTopology: true
//useCreateIndex: true,
//useFindAndModify: false
})
.then(() => {
console.log('DB connection successful!')
})
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
(base) praveenkumarsiddela@Praveens-MacBook-Pro Module 7 Project Assignment % node server.js
dirname /Users/praveenkumarsiddela/Downloads/Module 7 Project Assignment
views /Users/praveenkumarsiddela/Downloads/Module 7 Project Assignment/views
public /Users/praveenkumarsiddela/Downloads/Module 7 Project Assignment/public
mongodb+srv://praveenkumarsiddelaasu:Praveen5@m3a1.u0kjwhr.mongodb.net/?retryWrites=true&w=majority&appName=M3A1
App running on port 4000...
To test the IFT 458 REST App Click Or Type: http://localhost:4000...
DB connection successful!
OUTLINE
TIMELINE
DOCKER CONTAINERS
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMAGES
Ln 1, Col 1 Spaces: 2 UTF-8 LF JavaScript

Home Page:

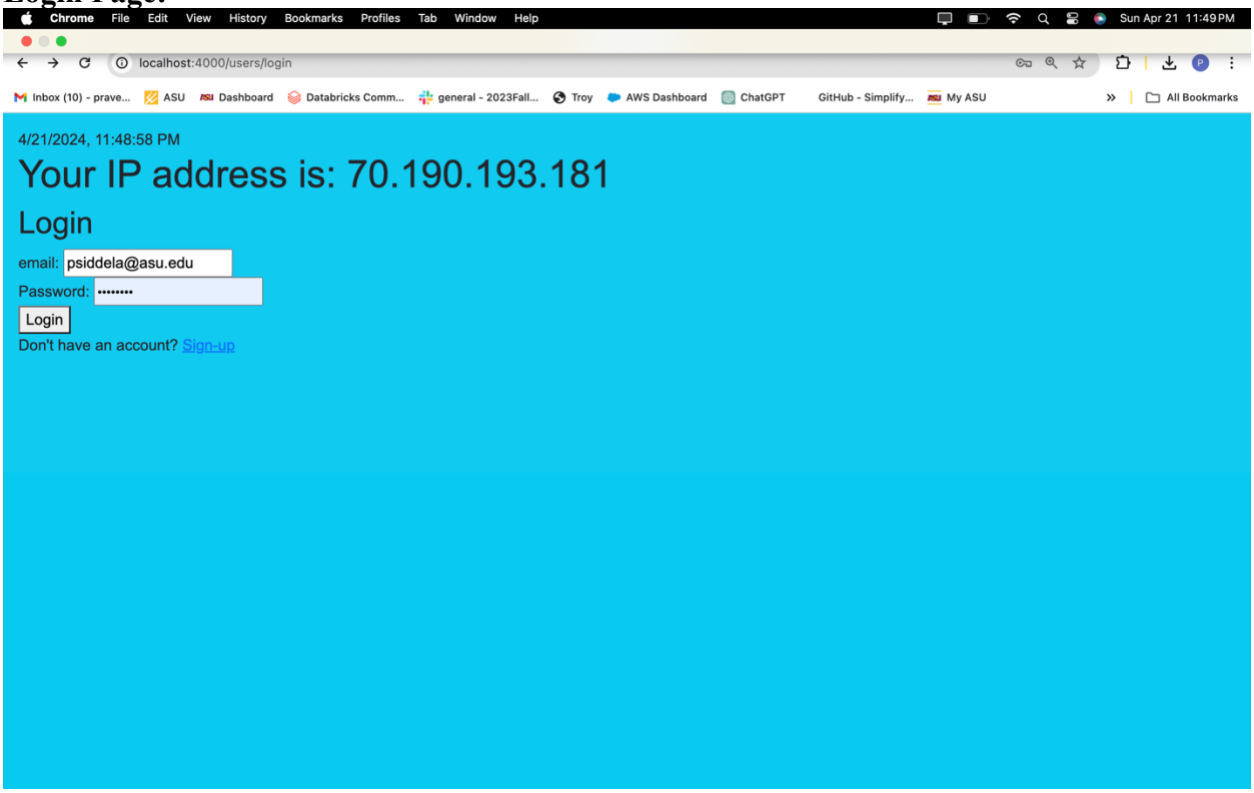

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 9:41 PM
localhost:4000
Inbox (10) - prave... ASU Dashboard Databricks Comm... general - 2023Fall... AWS Dashboard ChatGPT GitHub - Simplify... My ASU (51) Postings | Ha... All Bookmarks
Batch of IFT 458 Fall 2023
Server:
4/21/2024, 9:41:36 PM
Your IP address is: 70.190.193.181
Student Information
Student Image:
Student Name: Praveen Kumar Siddela
Student Email: psiddela@asu.edu
Student ID: 1227371004
Course #: IFT 554
Welcome to Book Exchange!
Book Exchange
Register
Login
Available Books:
Book Exchange Platform © IFT 510 - Fall 2023

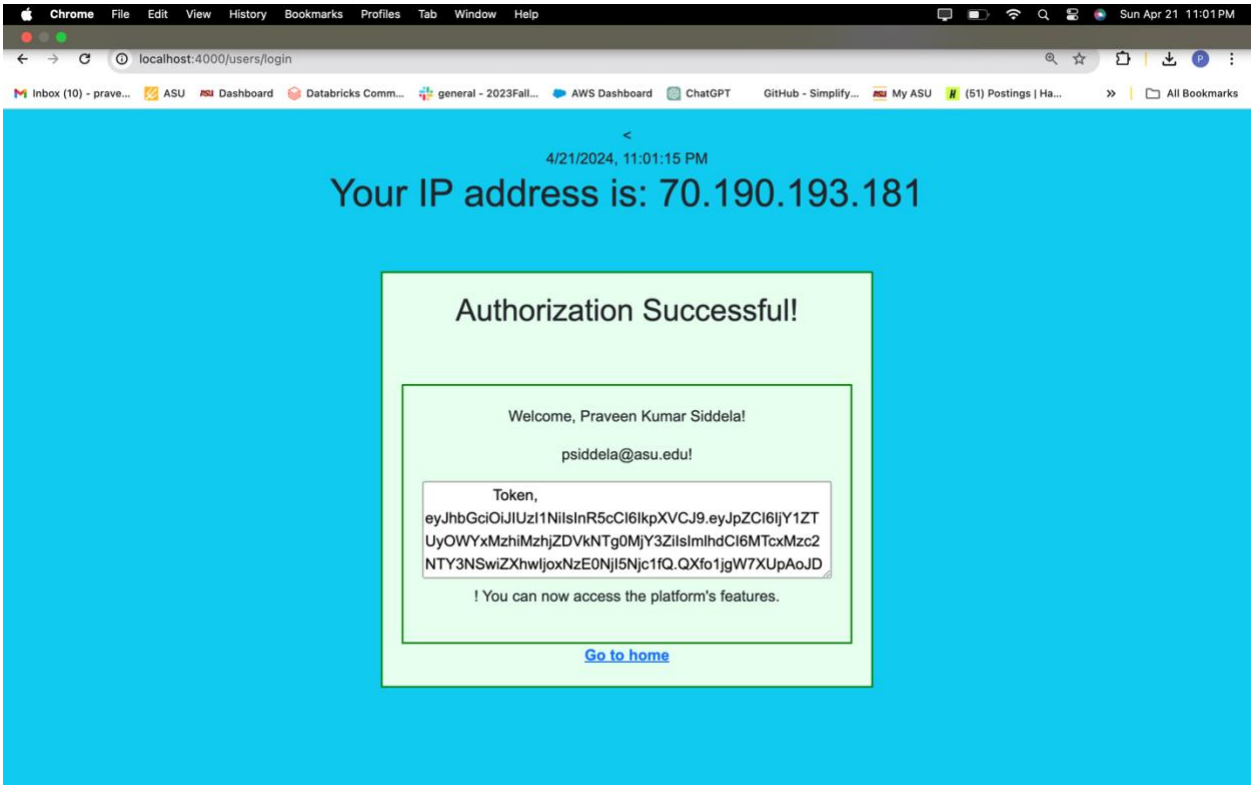
Signup:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 11:01 PM
localhost:4000/users/signup
4/21/2024, 11:01:24 PM
Your IP address is: 70.190.193.181
Register >
Name: Jeffrey
Email: Jeffrey@asu.edu
Password:
Confirm Password:
Signup
Already have an account? Login

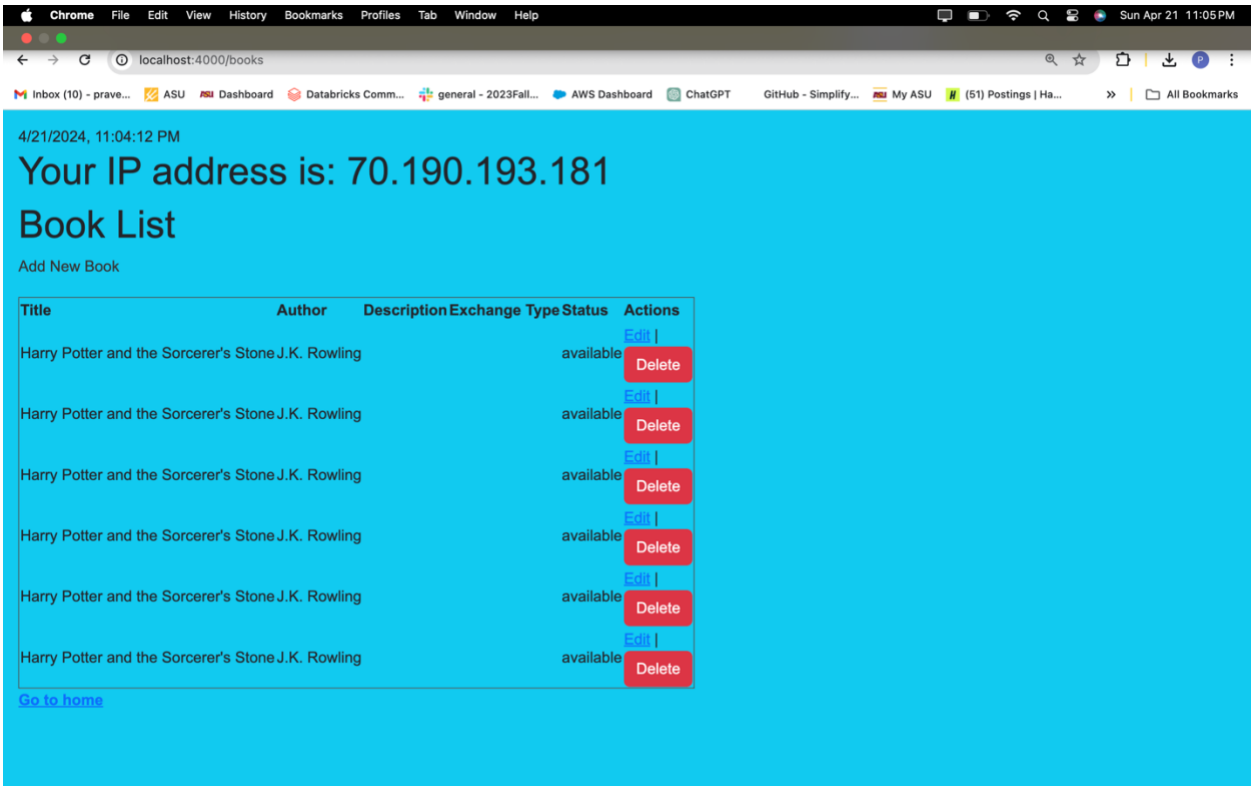

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 11:01 PM
localhost:4000/users/signup
4/21/2024, 11:01:50 PM
Your IP address is: 70.190.193.181
Authorization Successful!
Welcome, Jeffrey!
jeffrey@asu.edu!
Token,
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2MjV
mZDRlNzc4Yzc1ZjliMzlkNWJiNCIsImlhdCI6MTcxMzc2NTc
xMCwiZXhwIjoxNzE0NjI5NzEwfQ.FILLNXquSH-
! You can now access the platform's features.
Go to home

Login Page:

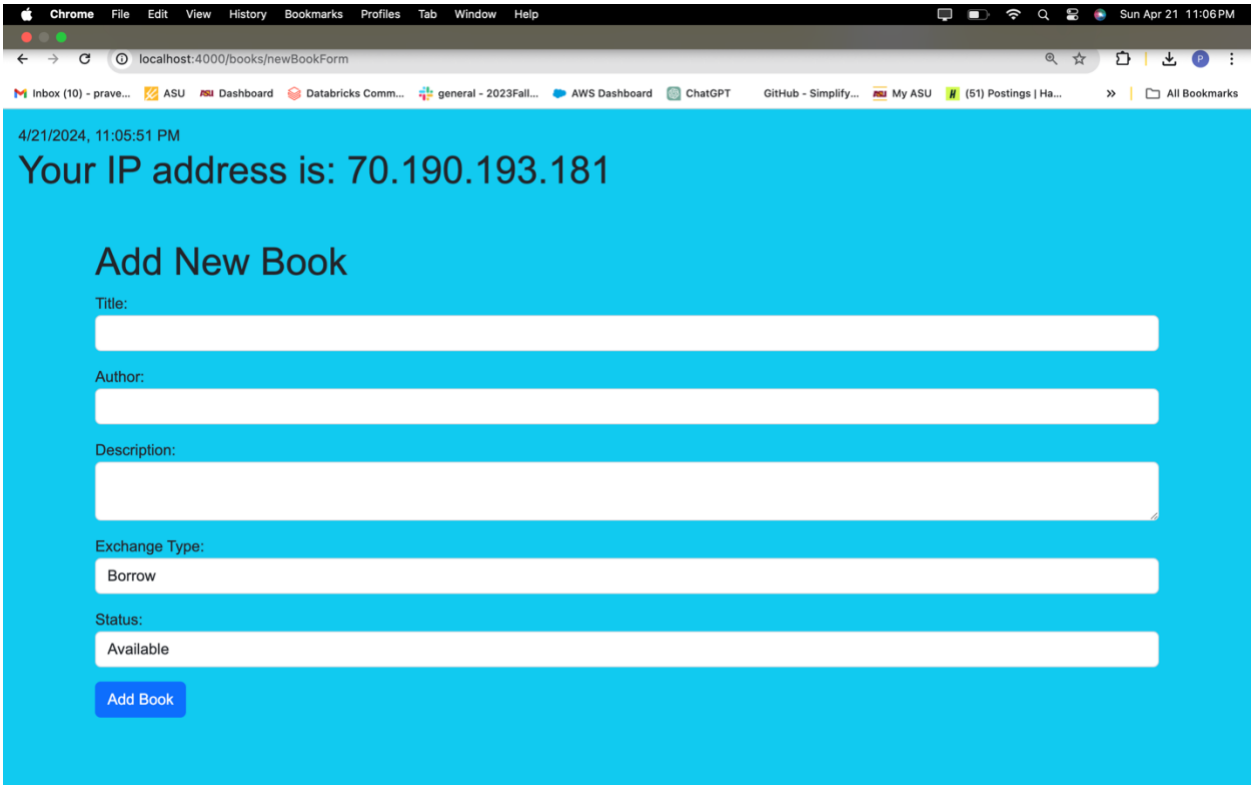


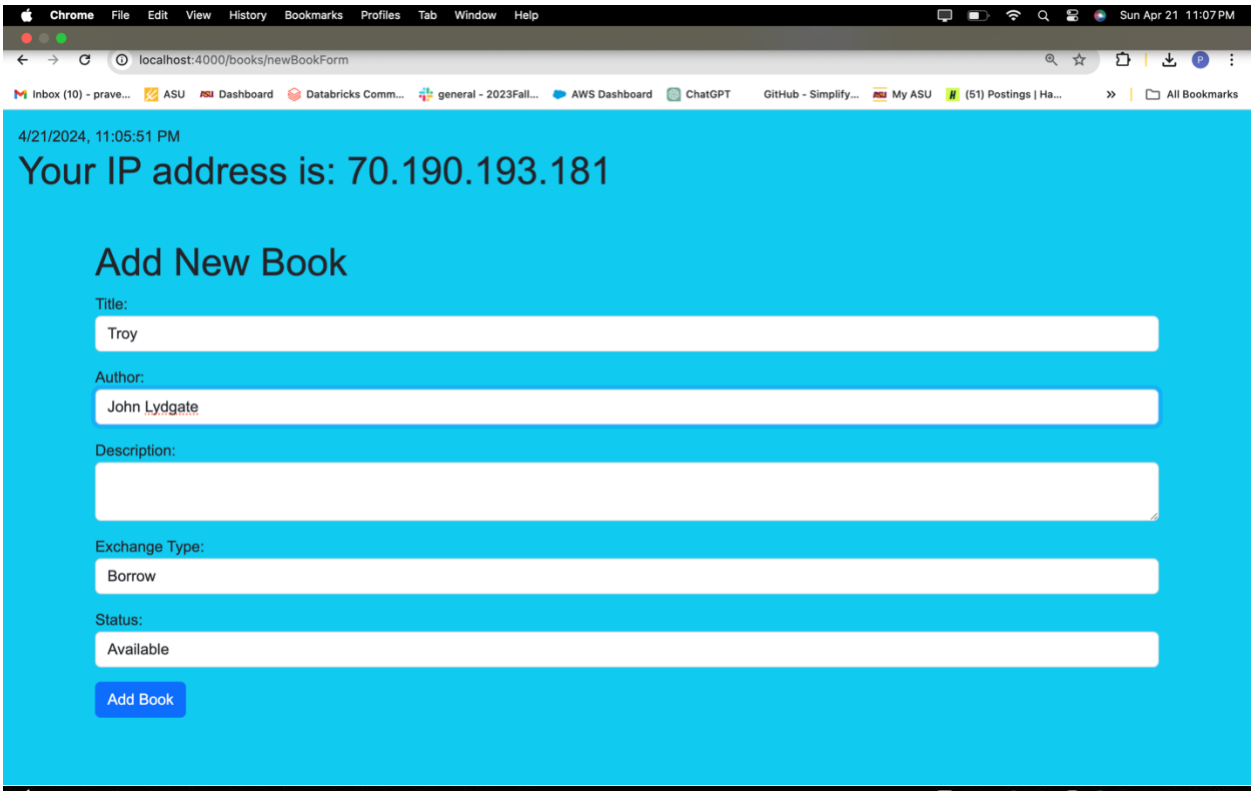


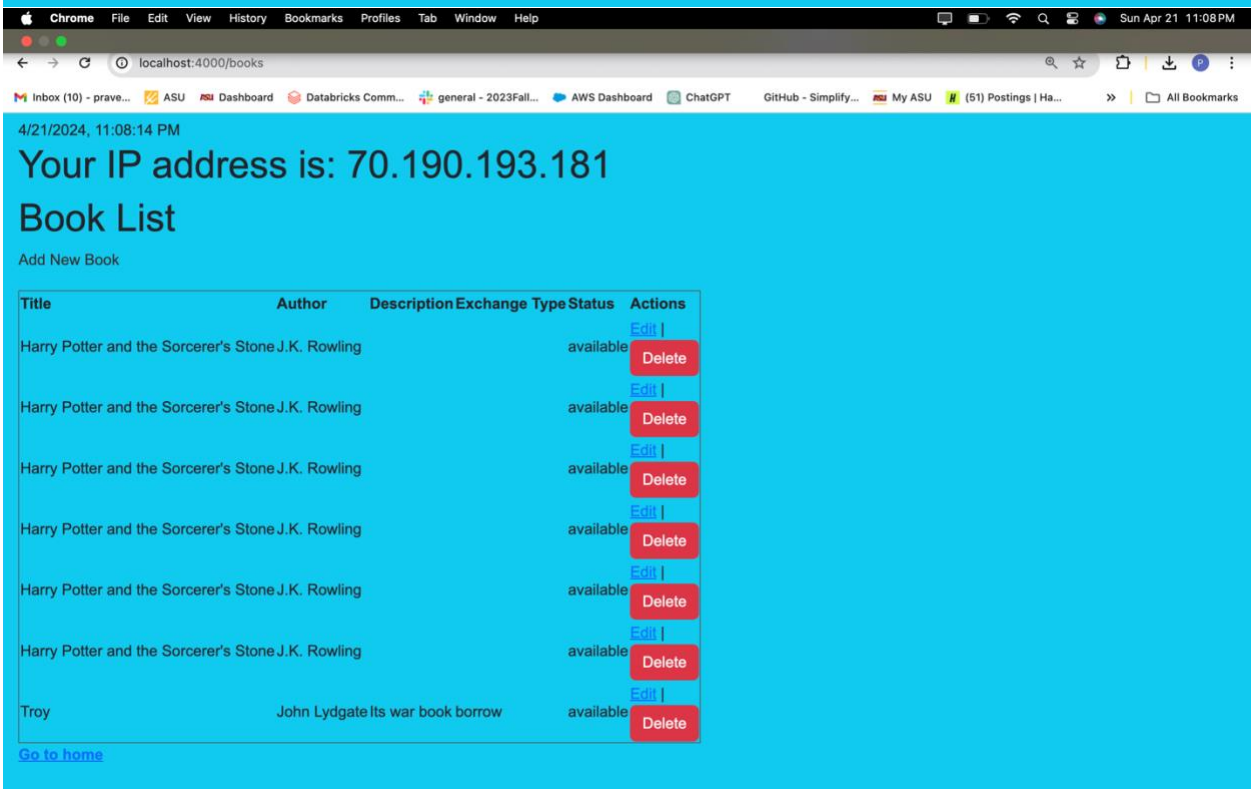
Retrieve All Books:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 11:05 PM
localhost:4000/books
4/21/2024, 11:04:12 PM
Your IP address is: 70.190.193.181
Book List
Add New Book
Title Author Description Exchange Type Status Actions
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Go to home

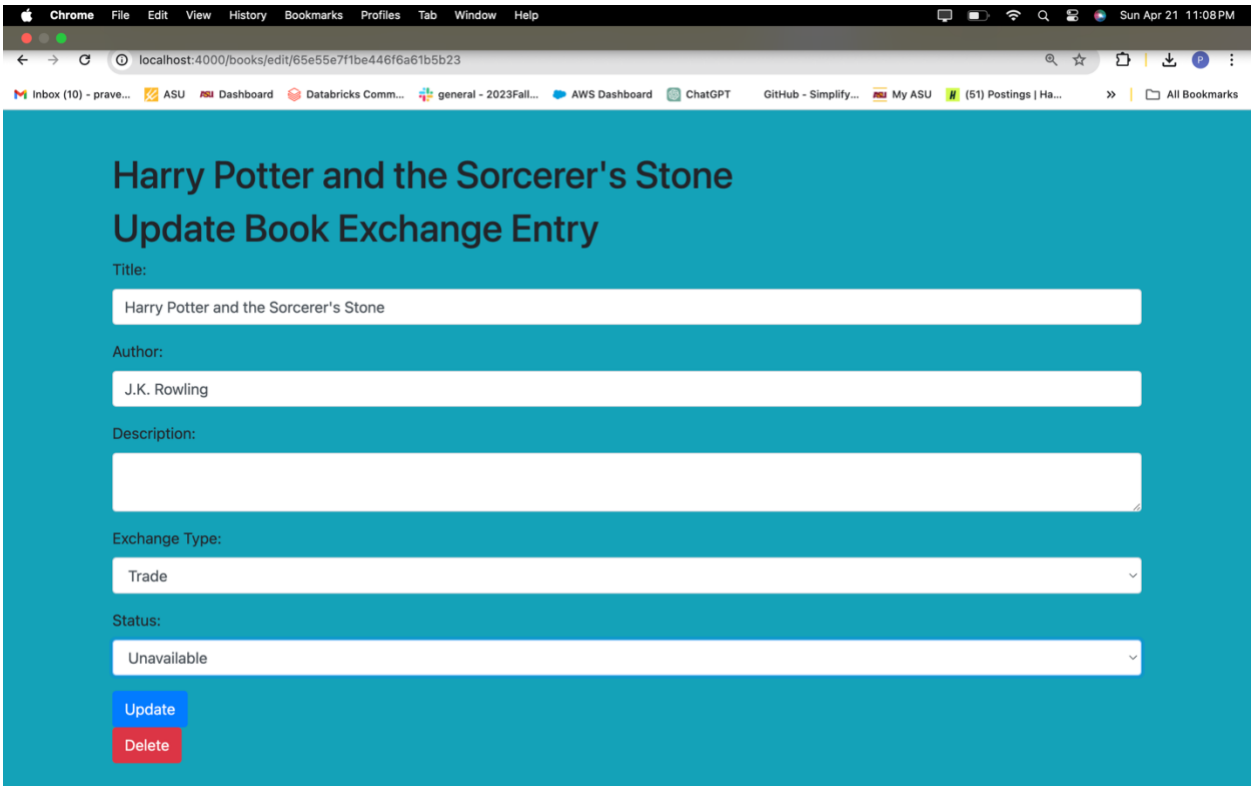
Add New Book:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 11:06 PM
localhost:4000/books/newBookForm
4/21/2024, 11:05:51 PM
Your IP address is: 70.190.193.181
Add New Book
Title:
Author:
Description:
Exchange Type:
Borrow
Status:
Available
Add Book

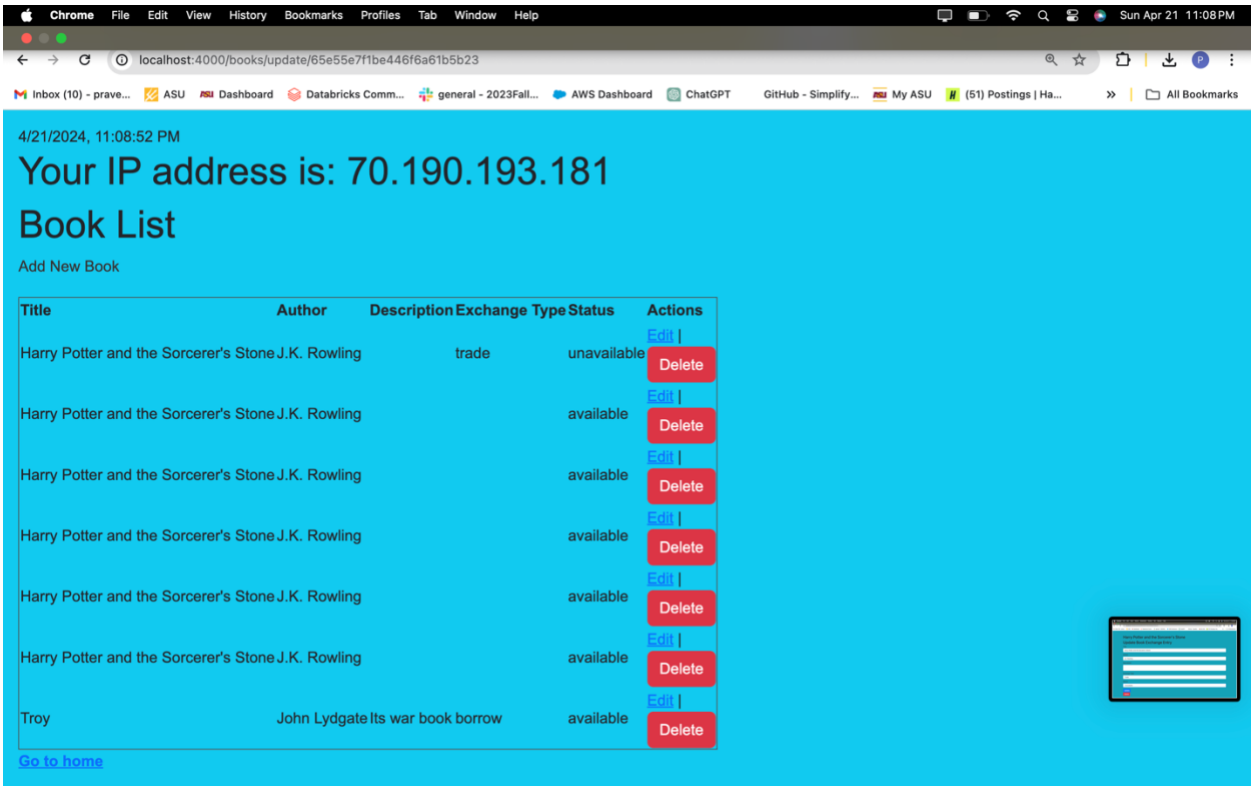

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 11:07 PM
localhost:4000/books/newBookForm
4/21/2024, 11:05:51 PM
Your IP address is: 70.190.193.181
Add New Book
Title:
Troy
Author:
John Lydgate
Description:
Exchange Type:
Borrow
Status:
Available
Add Book


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sun Apr 21 11:08 PM
localhost:4000/books
4/21/2024, 11:08:14 PM
Your IP address is: 70.190.193.181
Book List
Add New Book
Title Author Description Exchange Type Status Actions
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Harry Potter and the Sorcerer's Stone J.K. Rowling available Edit | Delete
Troy John Lydgate Its war book borrow available Edit | Delete
Go to home

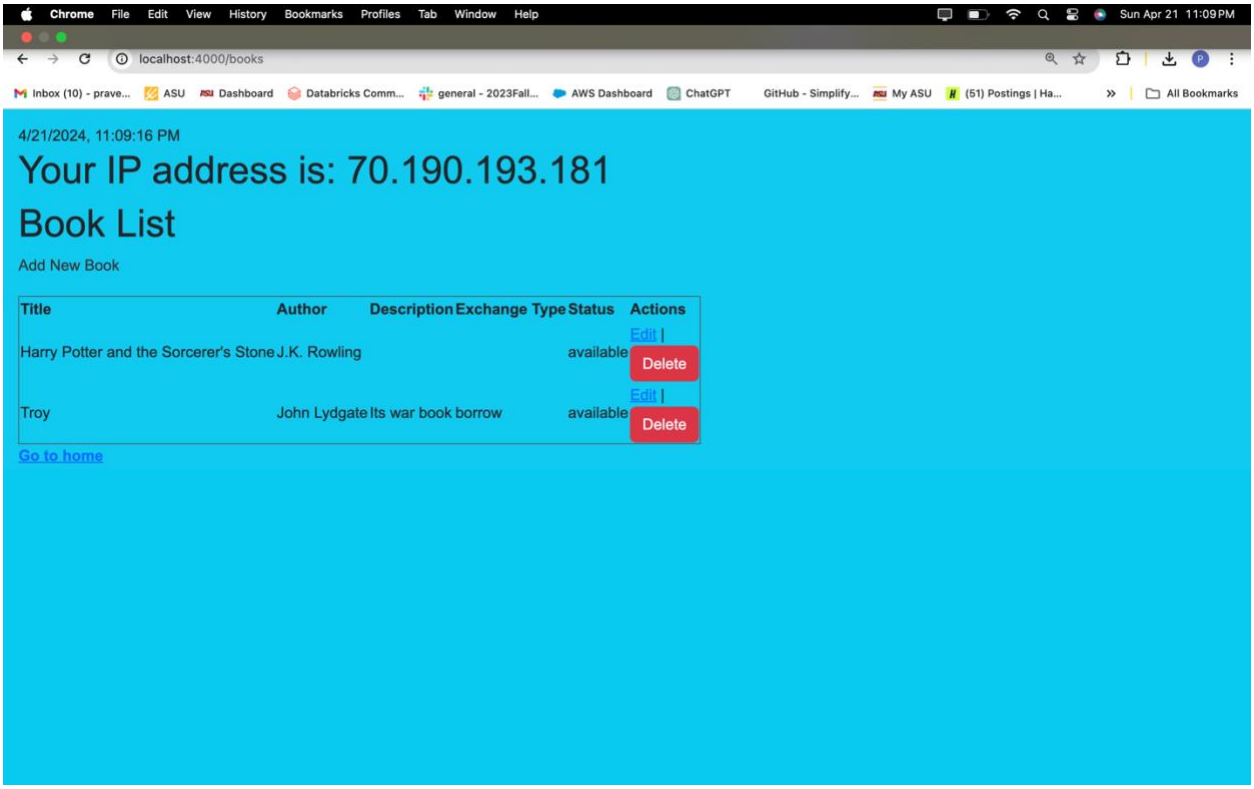
Edit of the books:



Deletion of Books:



After Deletion:

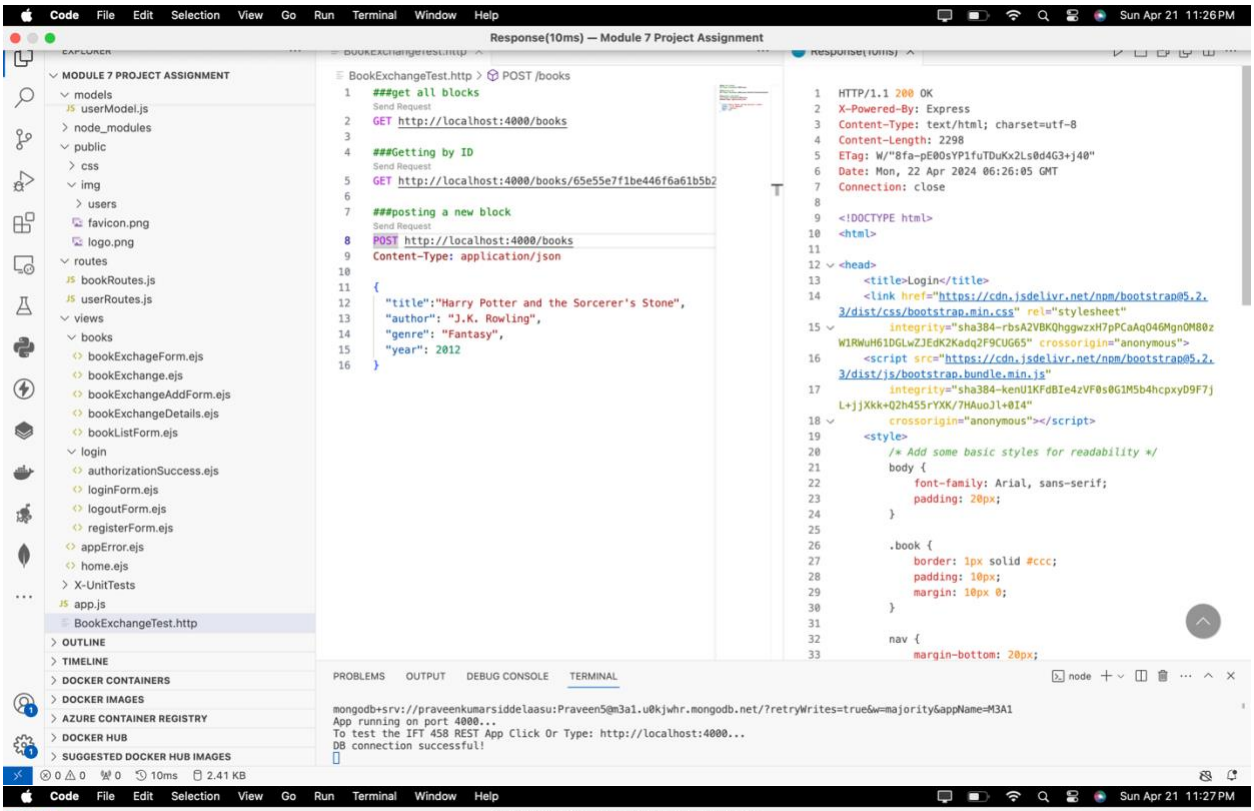


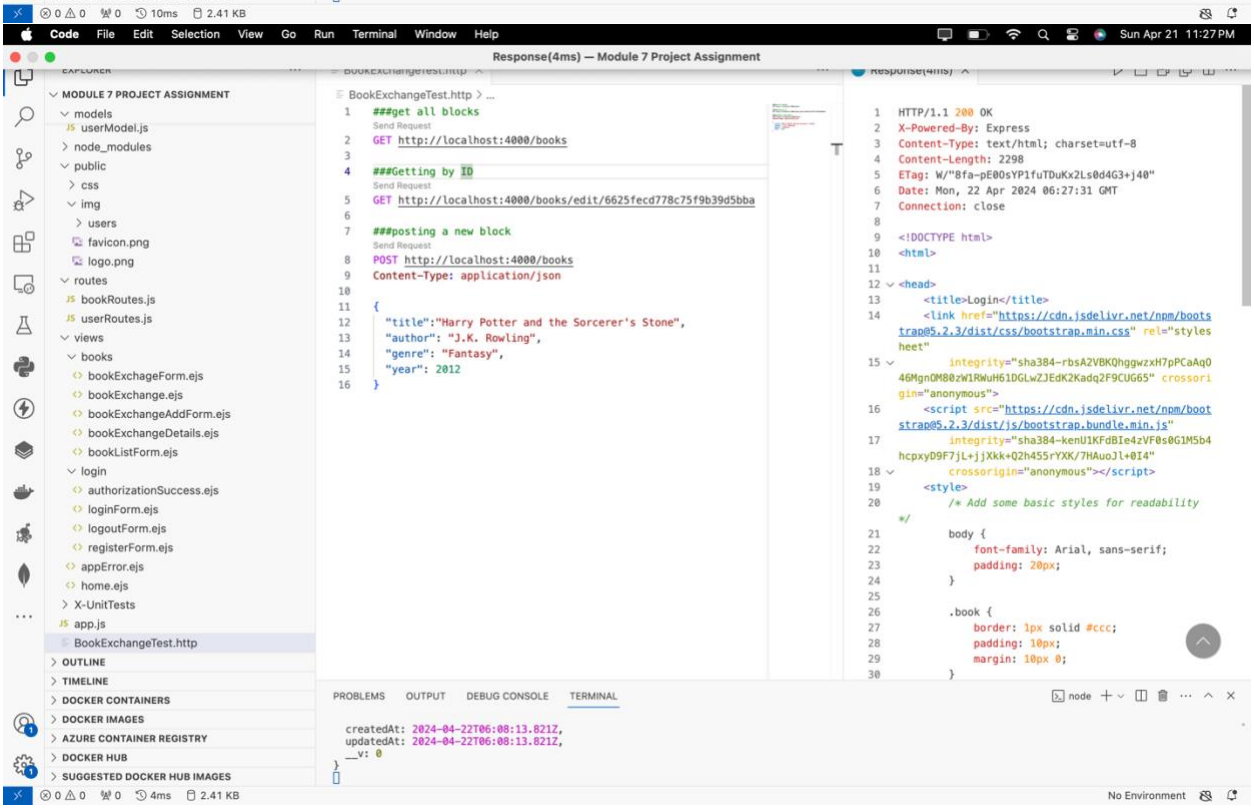
Screenshot 9: Of the testing process and responses for different API endpoints.

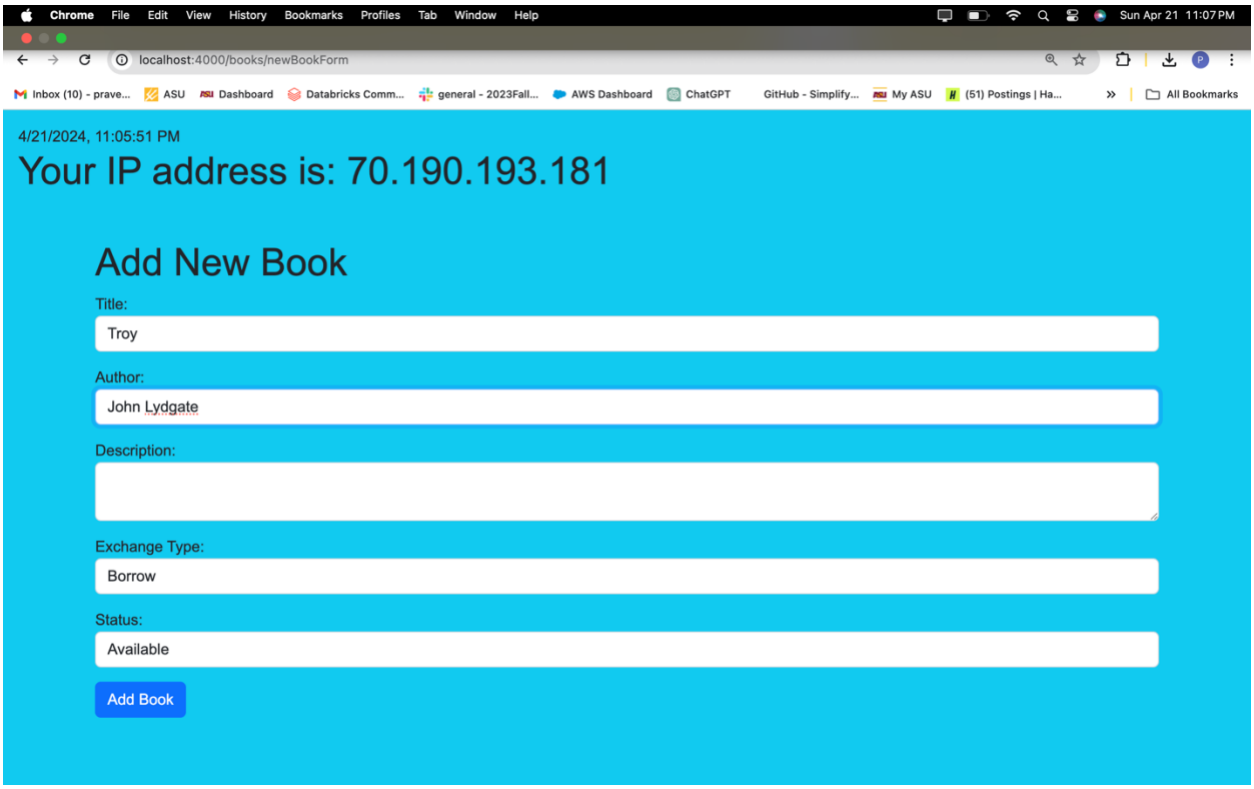
Feature	Method	URL End Point (the following are just the sample) You must update the URLs with the correct endpoints)
Register / Sign up	POST	https://localhost:4000/users/signup
Login	POST	https://localhost:4000/users/login
Protected Resource	GET	https://localhost:4000/books
View All Books	GET	https://localhost:4000/books
Add Book Exchange	POST	https://localhost:4000/books/newBookForm
Update Delete Exchange	POST	http://localhost:4000/books/edit/65e565c188ab31ac49627500

Screenshot 10: Of API specification documentation and screenshots of the corresponding code for each endpoint.

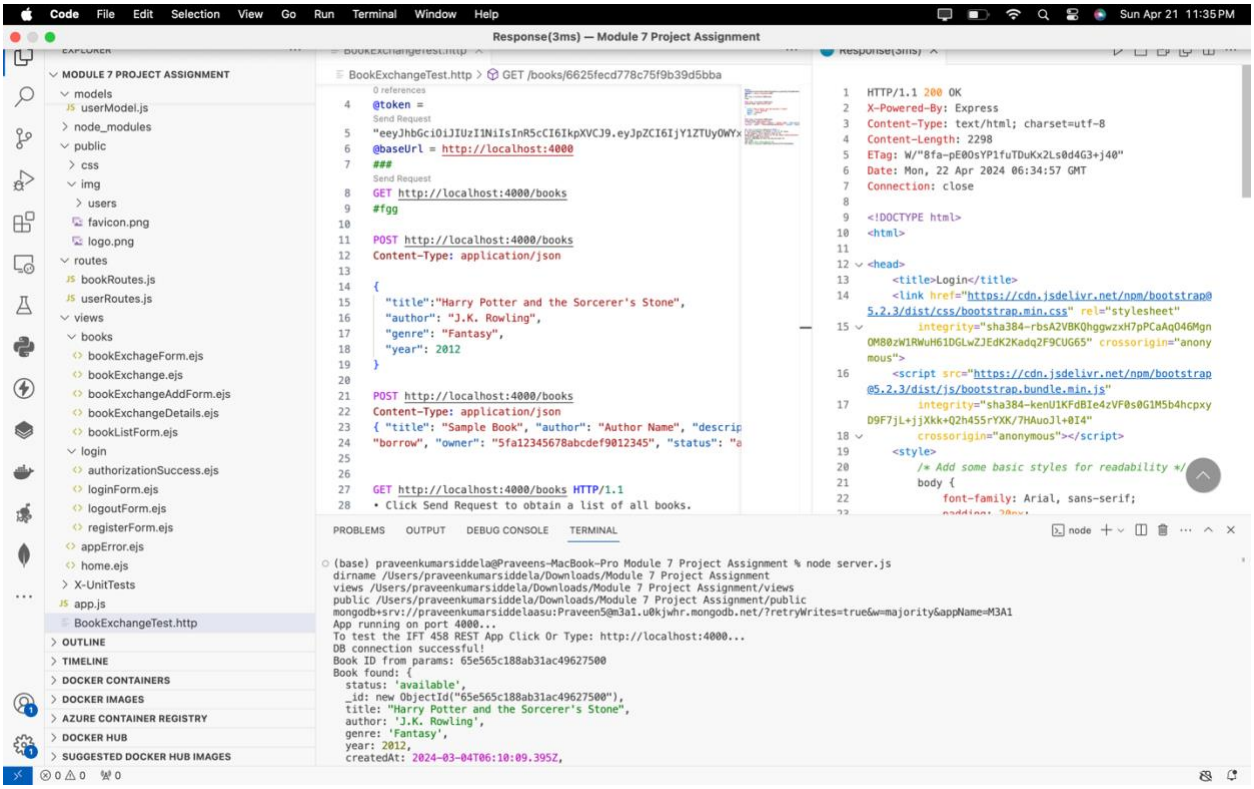
Post:Adding a New Book

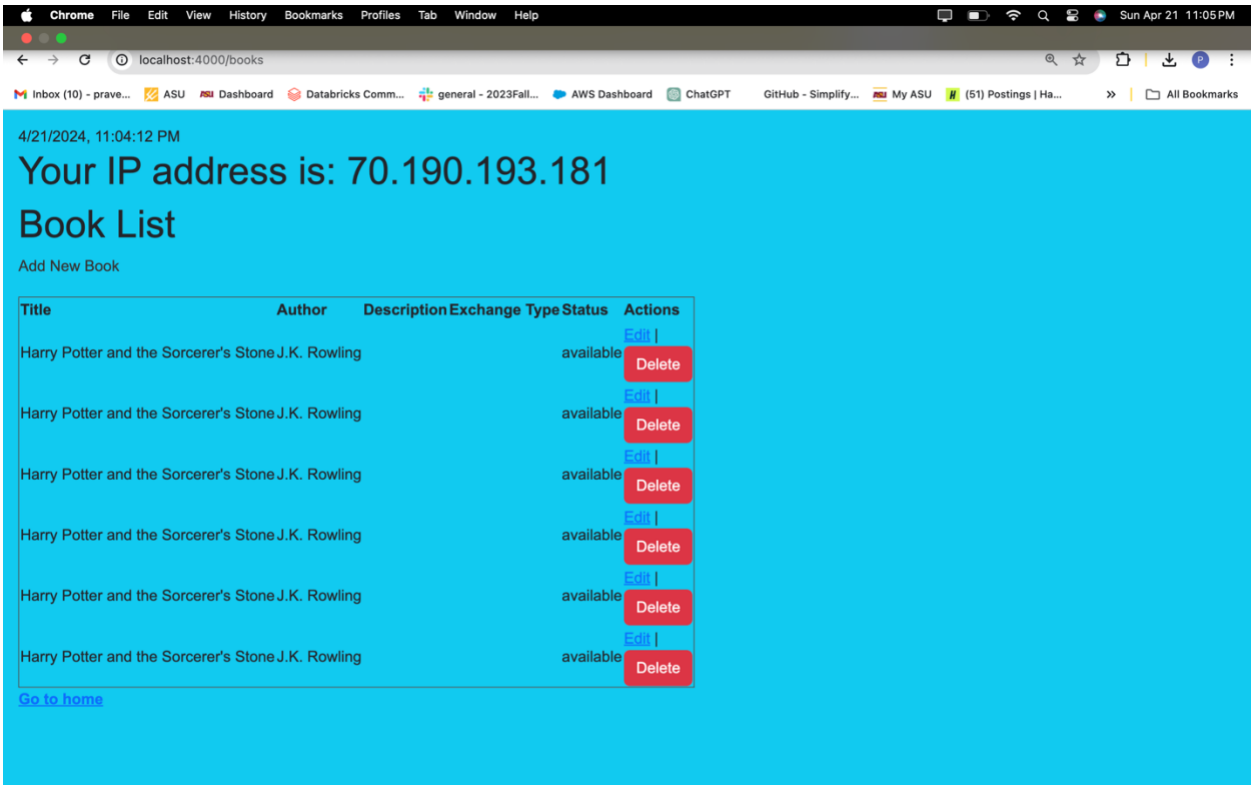

Response(10ms) — Module 7 Project Assignment
MODULE 7 PROJECT ASSIGNMENT
BookExchangeTest.http
###get all blocks
GET http://localhost:4000/books
###Getting by ID
GET http://localhost:4000/books/65e55e7f1be446f6a61b5b2
###posting a new block
POST http://localhost:4000/books
Content-Type: application/json
{
"title":"Harry Potter and the Sorcerer's Stone",
"author": "J.K. Rowling",
"genre": "Fantasy",
"year": 2012
}
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2298
ETag: W/"8fa-pE0OsYP1fuTDuKx2Ls0d4G3+j40"
Date: Mon, 22 Apr 2024 06:26:05 GMT
Connection: close
<!DOCTYPE html>
<html>
<head>
<title>Login</title>
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
mongodb+srv://praveenkumarsiddelaasu:Praveen5@m3a1.u0kjwhr.mongodb.net/?retryWrites=true&w=majority&appName=M3A1
App running on port 4000...
To test the IFT 458 REST App Click Or Type: http://localhost:4000...
DB connection successful!


Response(4ms) — Module 7 Project Assignment
MODULE 7 PROJECT ASSIGNMENT
BookExchangeTest.http
###get all blocks
GET http://localhost:4000/books
###Getting by ID
GET http://localhost:4000/books/edit/6625fecd778c75f9b39d5bba
###posting a new block
POST http://localhost:4000/books
Content-Type: application/json
{
"title":"Harry Potter and the Sorcerer's Stone",
"author": "J.K. Rowling",
"genre": "Fantasy",
"year": 2012
}
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2298
ETag: W/"8fa-pE0OsYP1fuTDuKx2Ls0d4G3+j40"
Date: Mon, 22 Apr 2024 06:27:31 GMT
Connection: close
<!DOCTYPE html>
<html>
<head>
<title>Login</title>
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
createdAt: 2024-04-22T06:08:13.821Z,
updatedAt: 2024-04-22T06:08:13.821Z,
__v: 0
}

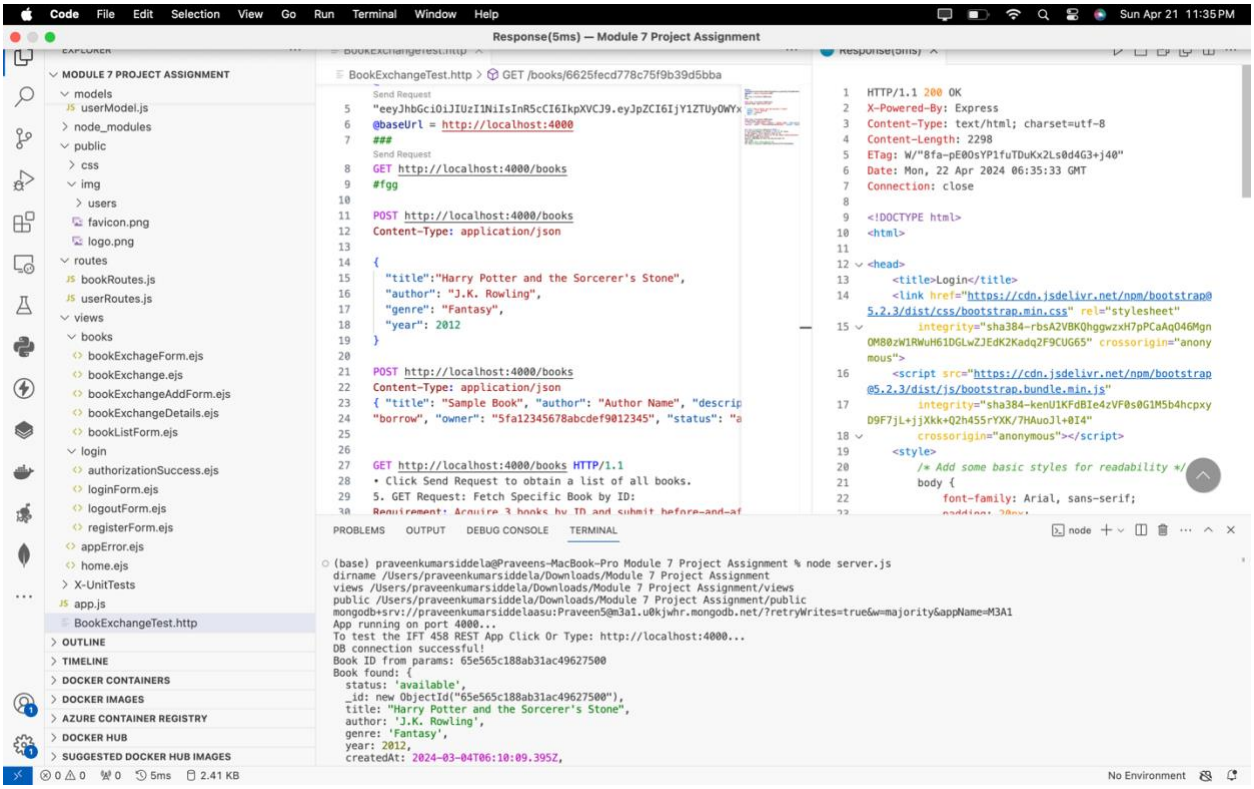


Get Request:Retrieve all Books

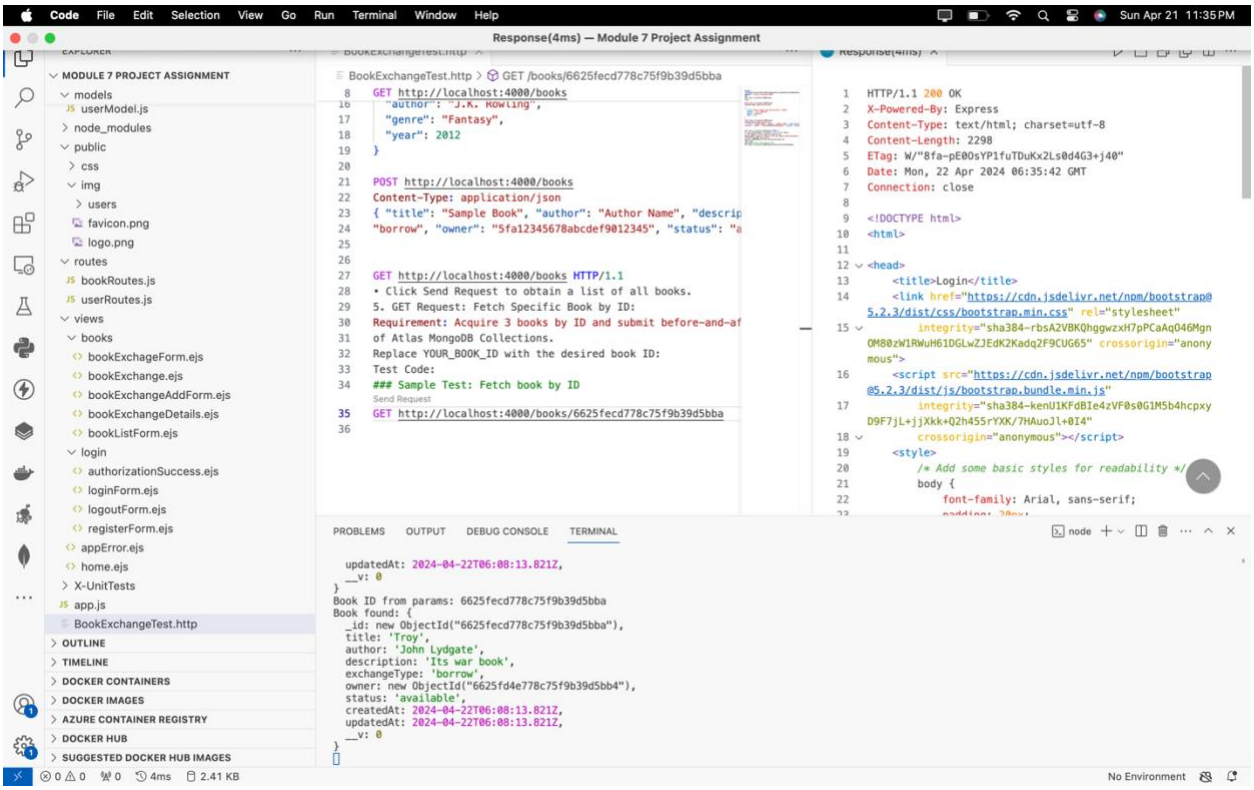




GET Request: Fetch Specific Book by ID:



PUT Request: Modify a Specific Book by ID:



Learning Outcomes of Module 4 to Module 7

Technical Learning Outcomes:

Module 4

In web applications, authentication and authorization are pivotal for ensuring secure user interactions. Authentication verifies users' identities using credentials like usernames and passwords, while authorization determines the actions users are allowed to take post-authentication. Session management is equally critical, as it maintains stateful information about user interactions, encompassing the creation, storage, and secure disposal of session data. Token-based authentication is a prevalent method, issuing tokens to authenticated users for subsequent authorization requests, with JSON Web Tokens (JWT) commonly employed due to their secure and efficient data transmission.

Testing RESTful APIs is imperative to guaranteeing their reliability and functionality. Integrated Development Environment (IDE) tools like Visual Studio Code offer REST client plugins, enabling direct API testing with simple syntax for interacting with various endpoints using standard HTTP methods. Additionally, debugging Node.js applications is vital for swiftly identifying and rectifying errors. IDEs provide debugging tools allowing developers to set breakpoints, pause code execution, and inspect variables and states, streamlining the debugging process for efficient troubleshooting. Protecting application resources involves implementing robust access controls and token expiry mechanisms, alongside session persistence techniques like cookies or local storage, ensuring secure user interactions and seamless experiences.

Module 5

Installing Git and Visual Studio Code is the first step in establishing a robust development environment, enabling version control and efficient project management. Cloning repositories allows developers to create local copies of remote codebases, fostering local development and facilitating collaboration with team members. Making changes to code, whether it involves editing existing files, adding new features, or removing unnecessary code, is essential for project progression and improvement. Staging and committing changes with descriptive messages ensures an organized version history, aiding in tracking project evolution and facilitating collaboration among team members.

Pushing changes to the remote repository enables synchronization between local and remote codebases, ensuring seamless collaboration and sharing of progress across the team. Branching out from the main codebase allows developers to isolate work on specific features or fixes, promoting development flexibility and minimizing conflicts. Middleware implementation in Node.js intercepts and processes incoming requests, facilitating tasks such as logging and authentication, thereby enhancing the functionality and security of web applications. Finally, opening pull requests initiates discussions, facilitates code review, and integrates proposed changes into the main repository, promoting collaboration and ensuring code quality and consistency across the project.

Module 6

Chained middleware enables the sequential execution of multiple middleware functions, allowing for modular and organized request processing in Node.js applications. Custom middleware development is crucial for addressing specific tasks such as logging and authentication, tailoring middleware functions to meet the application's unique requirements. Middleware error handling

plays a vital role in enhancing application reliability by implementing middleware to catch errors and send appropriate responses, ensuring smooth user experiences and preventing potential system failures. Verifying Docker, Node.js, and Visual Studio Code installations is essential for setting up the development environment, ensuring compatibility and readiness for containerization and application development processes. Writing a Docker file and building Docker images within Visual Studio Code streamlines the process of Docker image creation, facilitating efficient deployment and management of containerized applications. Running Docker containers and configuring settings allows developers to manage containerization effectively, ensuring optimal performance and resource utilization while accessing the deployed Node.js app.

Handling dynamic parameters in routes through dynamic routing mechanisms enhances the flexibility and scalability of web applications, accommodating varying user inputs and requirements. Middleware for request validation is crucial for ensuring data integrity and security by validating URL parameters and responding to errors appropriately, safeguarding against malicious inputs and potential vulnerabilities. Implementing complex middleware chains tailored to route-specific tasks allows for intricate request processing and handling in Node.js applications, enabling robust and efficient application behavior. Lastly, error handling middleware plays a crucial role in managing various error types, ensuring graceful degradation and consistent performance even in the face of unexpected issues, thereby enhancing the reliability and resilience of the application.

Module 7

To ensure secure communication in a Windows environment, OpenSSL installation is essential, providing a toolkit for SSL/TLS protocols. Generating a self-signed SSL certificate and private key using OpenSSL enables secure connections for applications. Configuring an HTTPS server in

a Node.js application with the generated SSL certificate and private key ensures encrypted communication, enhancing data security. Implementing HTTP status codes like 200, 201, 400, 404, and 500 allows for precise server responses, improving user experience and troubleshooting. Developing custom error handling middleware, including 404 (Not Found) and general server errors, ensures robust error management and graceful degradation. Using try/catch blocks with Promises facilitates graceful error handling during asynchronous operations, enhancing application reliability.

For a secure middleware and backend database setup in a Book Exchange Application, organizing the project directory structure according to the Model-View-Controller (MVC) pattern ensures a clear separation of concerns. Reviewing the book model schema and relationships for storing book information in the backend database provides insights into data organization. Implementing logic for CRUD operations related to books in the backend through the book controller enhances data management and manipulation capabilities. Defining routes for handling HTTP requests related to book resources establishes endpoints for CRUD operations, facilitating communication between the frontend and backend. Customizing view templates (*.ejs files) in the view folder personalizes the user interface and enables interaction with backend APIs. Configuring database connections, starting the server, and testing API functionality using tools like Postman or Thunder Client ensures proper application execution and functionality verification.

References

https://medium.com/@dushyanthak/best-practices-for-writing-custom-middlewares-in-asp-net-core-97b58c50cf9c

https://stackoverflow.com/questions/62398/what-are-the-best-practices-for-the-middleware-api

https://www.omnitas.com/how-to-choose-the-right-middleware-for-enterprise-integration/

Appendices

1. **Sample Code Snippets:**
    - Views (bookExchange.ejs)
    - Server (server.ejs)
    - Controller(booksController.js)
2. **Screenshots:**

- Updated view files (*.ejs) and changes made
- BookModel.js file and explanation
- Home page displayed in the browser
- Personalized user interface pages
- Testing process and responses for different API endpoints captured in Postman
- Project directory structure showcasing MVC organization
- Architecture diagram showcasing the components and their interactions
- Flowchart illustrating the flow of the Book Exchange Application
- Terminal showing successful installation of dependencies
- bookController.js with implemented logic
- bookRoutes.js file with defined routes

3. **Configuration Files:**

Configuration file (config.env) containing database connections and JWT parameters.
Server.js script setting up an HTTPS server in Node.js.

4. **API Specification:**

• API documentation detailing the endpoints, request methods, and expected data formats.

• Table presenting the Book Exchange API Specifications, containing endpoint specifics and descriptions gathered during testing.