Module 8: **Final Project

Title of the Project: Book Exchange Platform

Ganavi Hemachandra

Department of Information Technology, Arizona State University

IFT 458/554: Middleware Programming & Database Security

Prof. Dinesh Sthapit

December 01, 2023

Table of Contents

Abstract

The Book Exchange Project stands as a dynamic web application at the intersection of technology and literature, crafted with JavaScript, Node.js, and MongoDB. Tailored for book enthusiasts, the platform offers a seamless experience allowing users to effortlessly register, log in, and immerse themselves in a vibrant community of book exchange activities. Recognizing the paramount importance of user security, the implementation goes the extra mile in fortifying its architecture. Central to this is the integration of JSON Web Tokens (JWT), a cutting-edge technology that underpins user authentication and authorization processes. By employing JWT, the system establishes a robust and trusted framework for validating user identities, ensuring that only authorized individuals engage in the book exchange activities. This not only safeguards user data but also enhances the overall security posture of the platform. As users embark on their literary journeys within the Book Exchange Project, they can do so with confidence, knowing that the application's commitment to security creates a safe and trustworthy environment for the exchange of ideas and books alike.

Introduction

Overview of the Project

The Book Exchange Project is a dynamic and user-centric web application that seamlessly combines technology and literary passion. Developed with JavaScript, Node.js, and MongoDB, the platform serves as a virtual hub for book enthusiasts, offering a range of interactive features for users to register, log in, and actively participate in book exchange activities. The primary focus of the project is on providing a secure and trustworthy user experience.

Security is a top priority in the Book Exchange Project, and it is achieved through the implementation of advanced measures, notably the integration of JSON Web Tokens (JWT). This cutting-edge technology plays a pivotal role in ensuring robust user authentication and authorization processes. By leveraging JWT, the system establishes a reliable framework for validating user identities, enhancing the overall security of the platform.

Users engaging in the Book Exchange Project can enjoy a vibrant and secure community, where their literary pursuits are facilitated in a protected online environment. The combination of innovative technologies and a commitment to security creates a platform where book lovers can connect, exchange ideas, and share their passion for literature with confidence and peace of mind.

Problem Statement

Problem Statement: The project seeks to fulfill the demand for a highly user-friendly and secure platform tailored to the needs of avid readers, fostering connections, enabling seamless book exchanges, and facilitating efficient management of individual book collections. The challenges at hand encompass the imperative of implementing robust security measures for user authentication, the seamless execution of CRUD (Create, Read, Update, Delete) operations to streamline book management, and the imperative to deliver a responsive and intuitively designed user interface. Addressing these challenges is crucial to creating an inclusive and enjoyable digital space that not only ensures the privacy and security of users but also enhances the overall experience, making book exploration and collection management a delightful and effortless endeavor.

Scope and Limitations:

The Book Exchange Project ambitiously aims to establish a comprehensive online platform dedicated to the diverse needs of book enthusiasts. The primary objective is to provide users with a seamless and user-friendly interface, fostering connections and enabling the exchange and management of book collections. Within the project's scope is the implementation of robust security measures, including the integration of technologies like JSON Web Tokens (JWT) to ensure secure user authentication. Furthermore, the project encompasses CRUD operations, allowing users to effortlessly create, read, update, and delete entries in their book collections. A pivotal aspect of the scope is the commitment to delivering a responsive and intuitively designed user interface, enhancing the overall experience for users engaged in book exploration and collection management.

Dependencies on external technologies, such as JSON Web Tokens (JWT) and database systems, may introduce challenges in terms of maintenance, updates, and potential compatibility issues. Adapting to technological advancements and ensuring seamless integration is crucial for long-term sustainability.

Justification of the Technology Stack:

The technology stack chosen for the Book Exchange Project, comprising JavaScript, Node.js, and MongoDB, is justified by several key considerations:

1. JavaScript for Frontend Consistency: Utilizing JavaScript for both frontend and backend development ensures consistency in the programming language across the entire application. This streamlines development, reduces context-switching for developers, and facilitates code reuse.

2. Node.js for Scalability: Node.js is chosen for the backend due to its event-driven, non-blocking architecture, which enhances scalability. It allows the server to handle many concurrent connections efficiently, making it well-suited for applications with potentially high traffic, such as a book exchange platform.

3. MongoDB for Flexible Data Modeling: MongoDB, a NoSQL database, is preferred for its flexibility in data modeling. As book-related data may vary in structure and metadata, a document-oriented database like MongoDB allows for easy adaptation to evolving data requirements without a rigid schema.

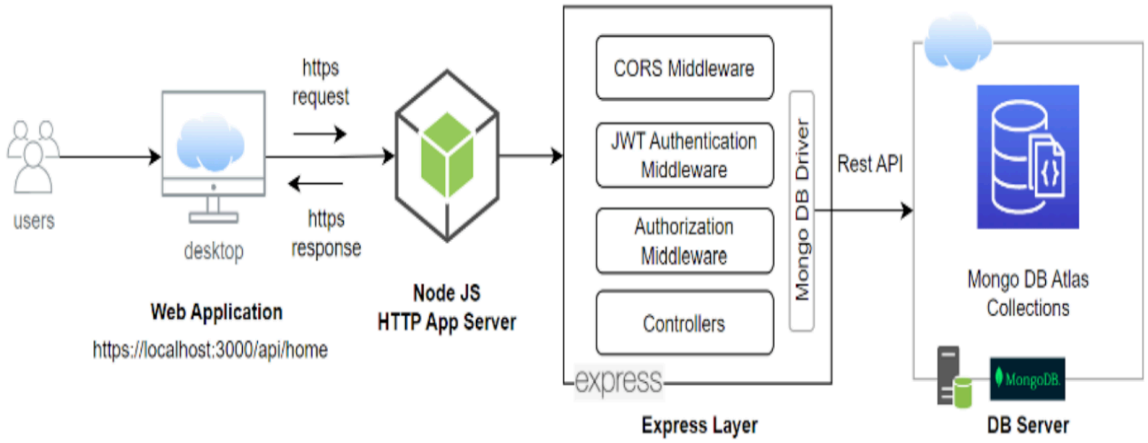
4. JSON Web Tokens (JWT) for Secure Authentication: The inclusion of JSON Web Tokens enhances the security of user authentication and authorization processes. JWT provides a stateless and secure way to transmit user information between the client and server, reducing the risk of session-related vulnerabilities.

5. RESTful API Design: The technology stack supports a RESTful architecture, allowing for clear and standardized communication between components. This design choice enhances interoperability, simplifies maintenance, and supports the gradual expansion of the project's features.

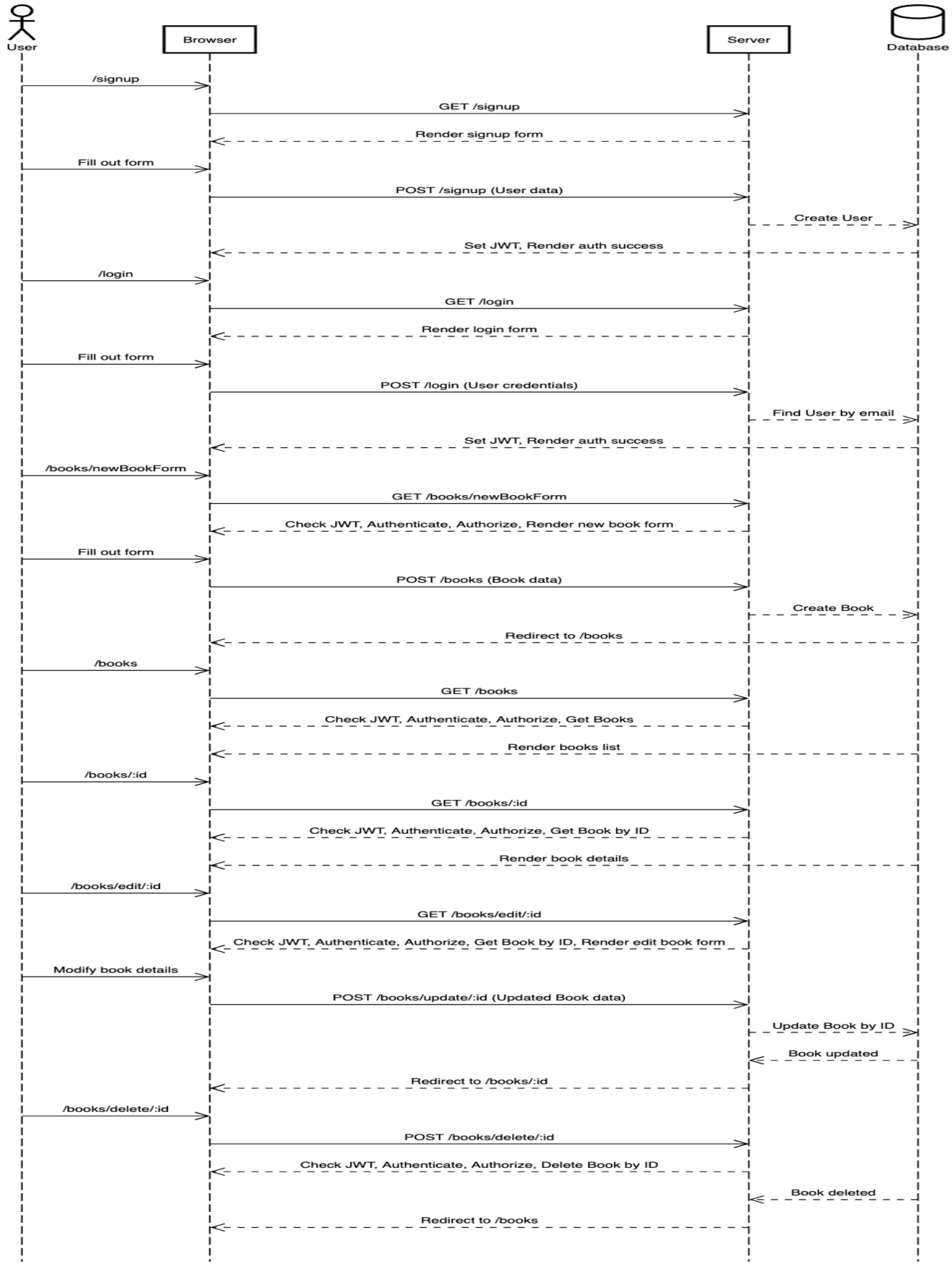
6. Open-Source Ecosystem: The chosen technologies are widely adopted and supported by vibrant open-source communities. This ensures access to a rich ecosystem of libraries, modules, and community-driven resources, facilitating development, troubleshooting, and future enhancements.

Project Design

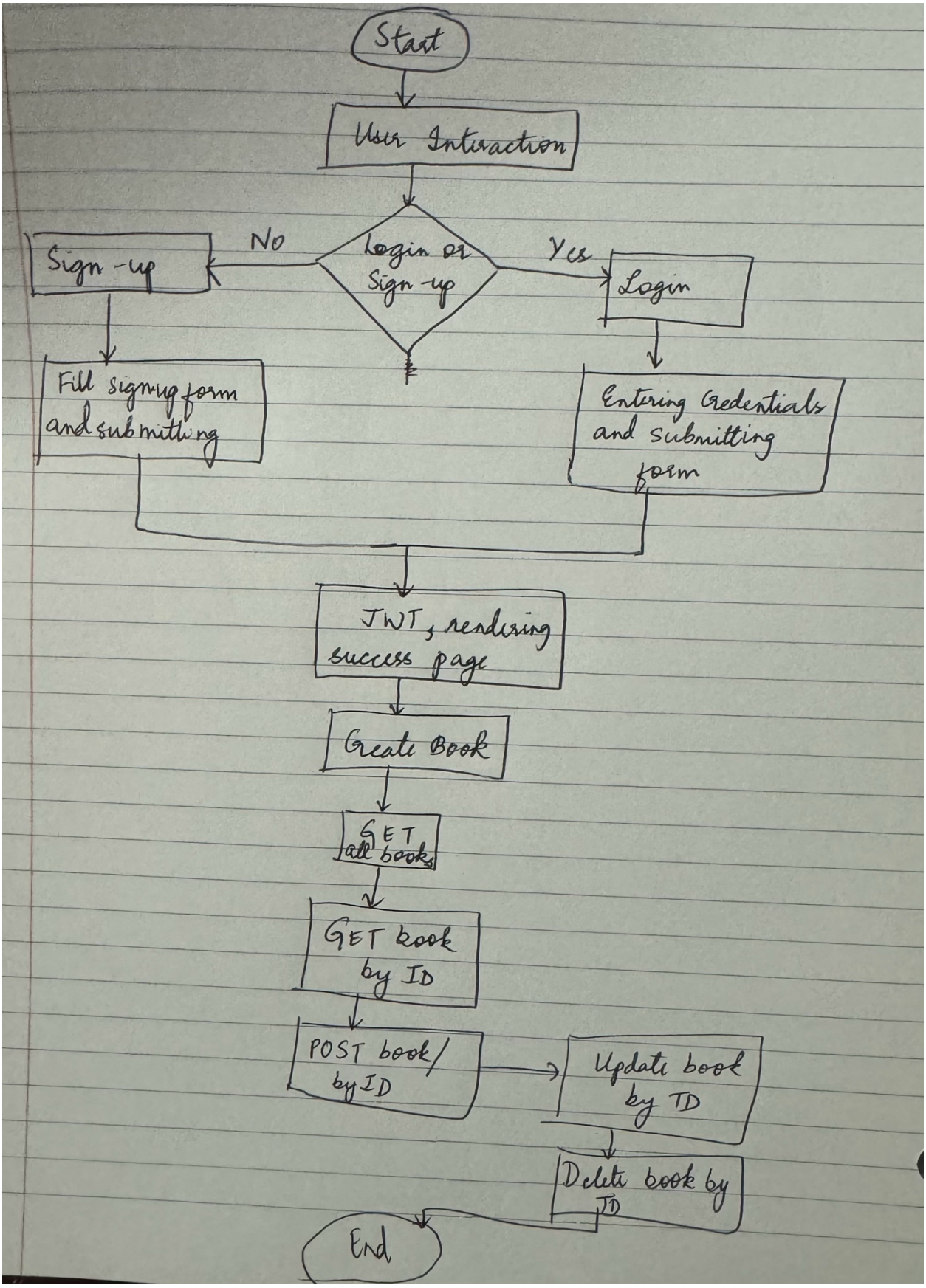
Architecture Diagrams:



Sequence Diagram:



Flow Chart:


Start
User Interaction
Login or Sign-up
No
Yes
Sign-up
Login
Fill signup form and submitting
Entering Credentials and submitting form
JWT, rendering success page
Create Book
GET all books
GET book by ID
POST book/ by ID
Update book by ID
Delete book by ID
End

Pseudocode or Code snippets:

1. **User Authentication & Authorization:**

```
function signup(userInput):
    try:
        newUser = createUser(userInput)
        createAndSendToken(newUser)
    catch (error):
        renderErrorPage(400, error.message)

function createAndSendToken(user):
    token = signToken(user._id)
    setCookie('jwt', token, cookieOptions)
    user.password = undefined
    renderAuthorizationSuccessPage(user, token)

function login(credentials):
    try:
        user = findUserByEmail(credentials.email)
        if (userExists(user) && isPasswordMatch(credentials.password, user.password)):
            createAndSendToken(user)
        else:
            renderErrorPage(401, 'Incorrect email and password!')
    catch (error):
        renderErrorPage(401, 'Incorrect email and password!')

function protect():
    token = extractTokenFromRequestHeader()
    decoded = verifyToken(token)
    currentUser = findUserById(decoded.id)
    if (userExists(currentUser)):
```

```
        grantAccessToProtectedRoute(currentUser)
    else:
        renderErrorPage(401, 'User not found!')
```

2. Book management:

```
function addBookForm():
    try:
        renderBookExchangeAddForm()
    catch (error):
        renderErrorPage(400, error.errors)

function createBook(bookData, currentUser):
    try:
        book = new Book(bookData)
        book.owner = currentUser._id
        newBook = saveBook(book)
        redirectToBooksList()
    catch (error):
        renderErrorPage(400, error.errors)

function getBooks():
    try:
        books = getAllBooks()
        renderBookListForm(books)
    catch (error):
        renderErrorPage(500, error.message)

function getBookById(bookId):
    try:
        book = findBookById(bookId)
        if (bookExists(book)):
            renderBookExchangeDetails(book)
        else:
```

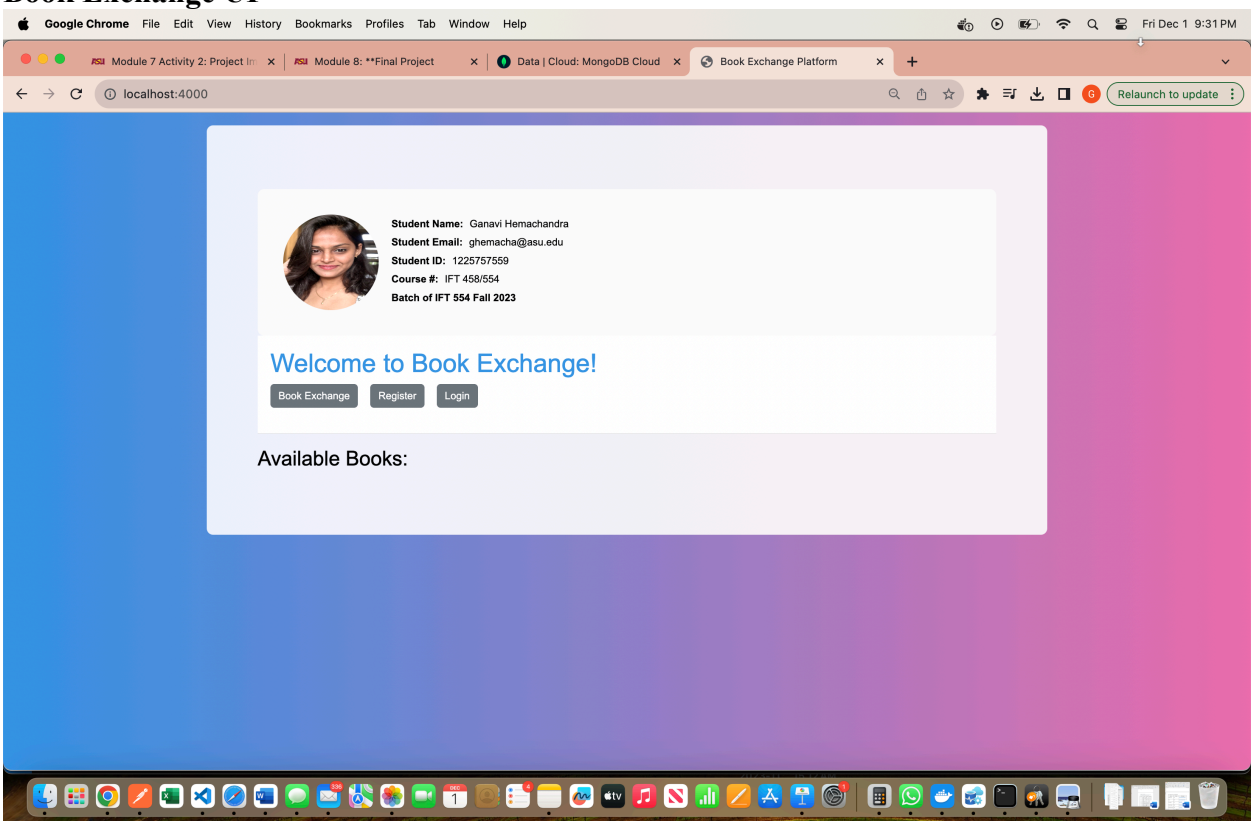
```
     renderErrorPage(404, 'Book not found')
    catch (error):
        renderErrorPage(500, error.message)

function updateBookForm(bookId):
    try:
        book = findBookById(bookId)
        if (bookExists(book)):
            renderBookExchangeDetails(book)
        else:
            renderErrorPage(404, 'Book not found')
    catch (error):
        renderErrorPage(400, error.errors)

function updateBookById(bookId, updatedBookData):
    try:
        updatedBook = updateBook(bookId, updatedBookData)
        redirectToBooksList()
    catch (error):
        renderErrorPage(500, error.message)

function deleteBookById(bookId):
    try:
        deletedBook = deleteBook(bookId)
        redirectToBooksList()
    catch (error):
        renderErrorPage(500, error.message)
```

API Specifications:

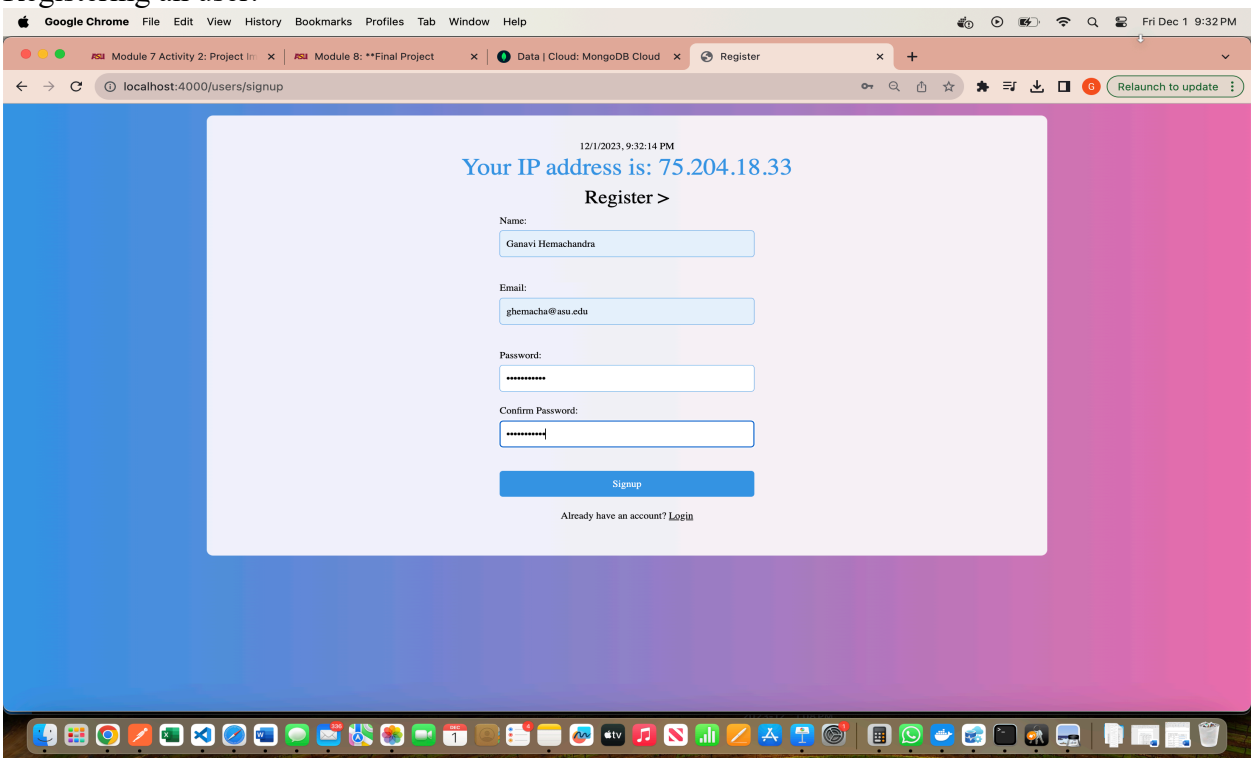
Feature	**Method**	**URL Endpoint**
Register/ Sign up	POST	http://localhost:4000/users/signup
Login	POST	http://localhost:4000/users/login
View All Books	GET	http://localhost:4000/books
Add Book Exchange	POST	http://localhost:4000/books
Update Book Exchange	POST	http://localhost:4000/books/update/:id
Delete Book Exchange	POST	http://localhost:4000/books/delete/:id

Screenshots

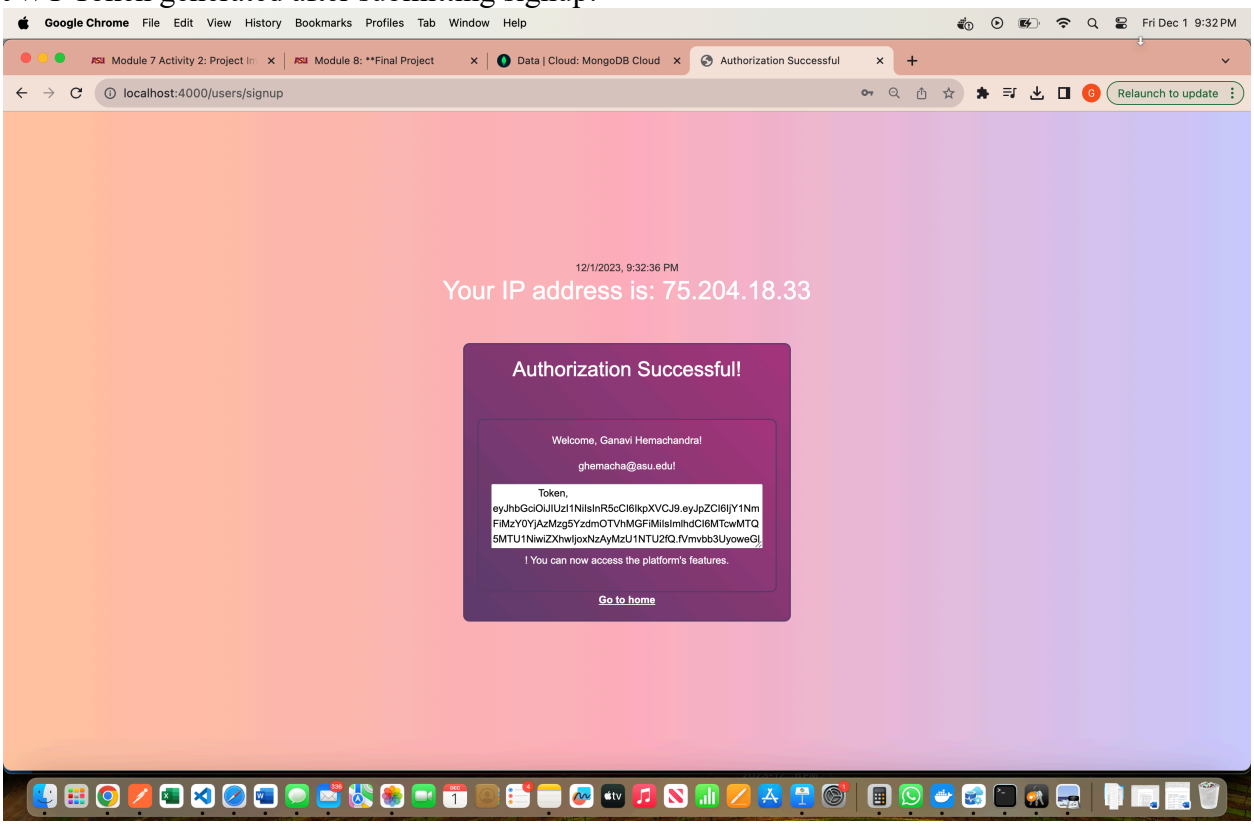
Book Exchange UI



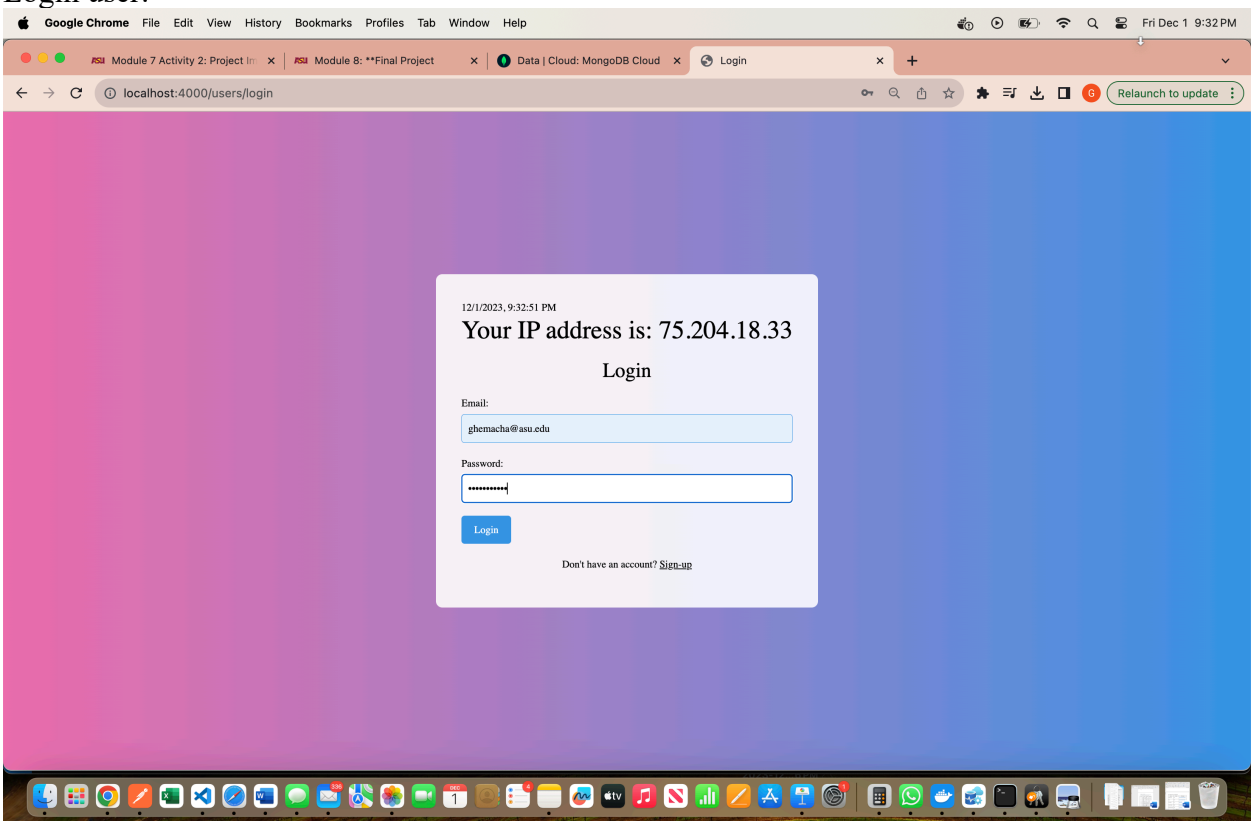
Registering an user:



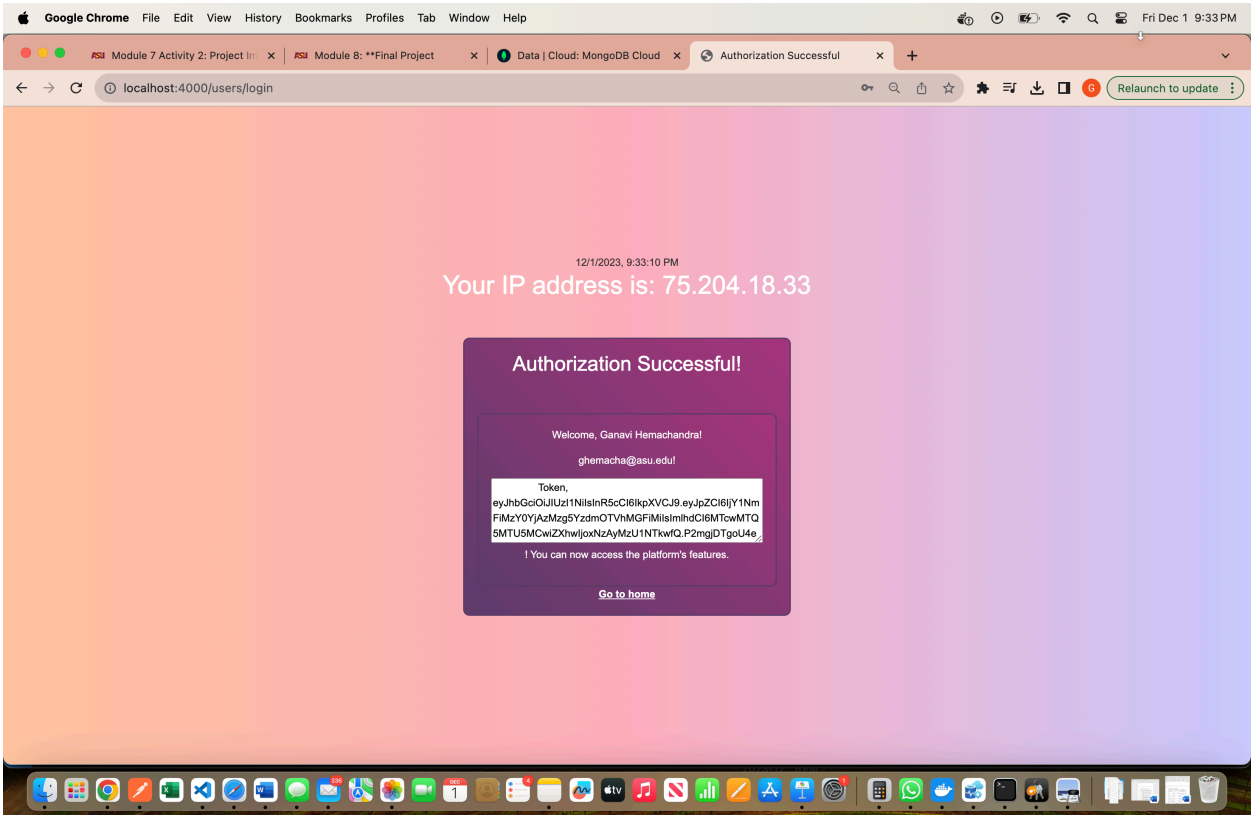
JWT Token generated after submitting signup:



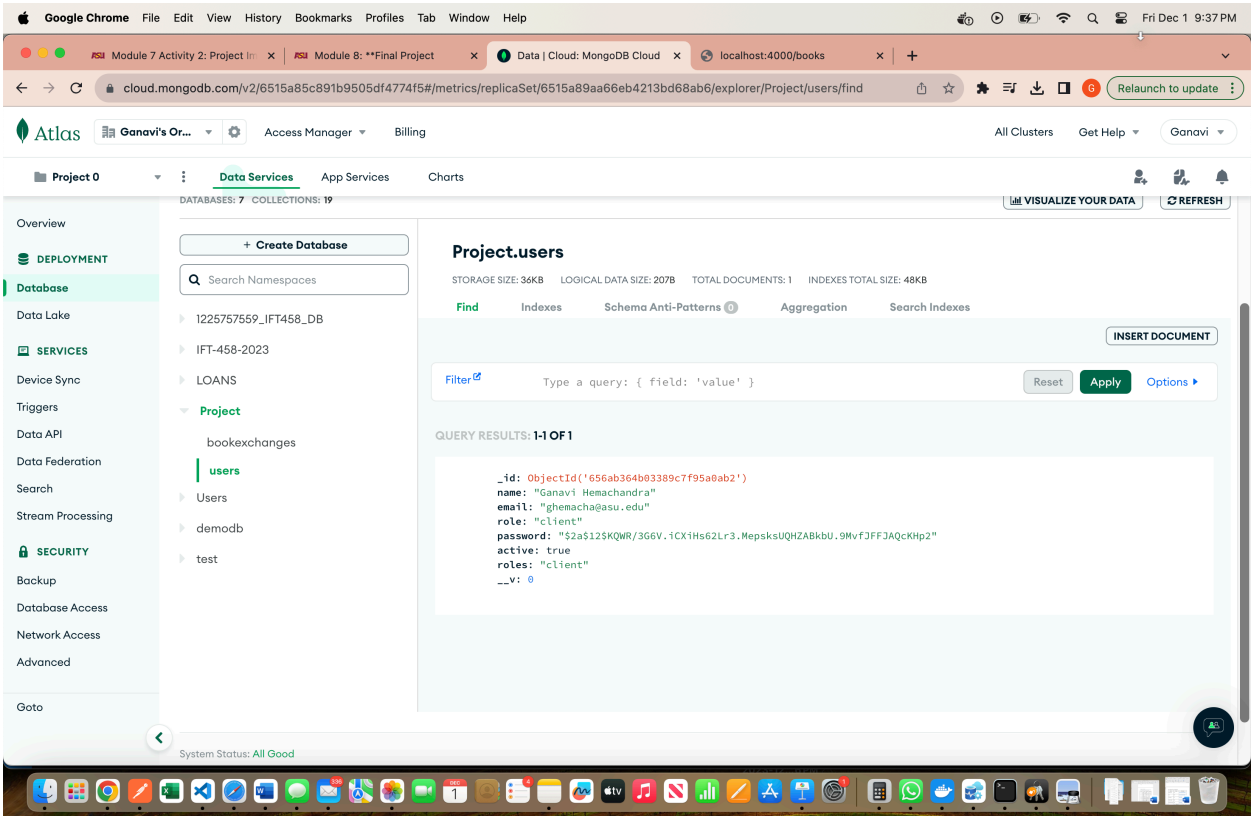
Login user:



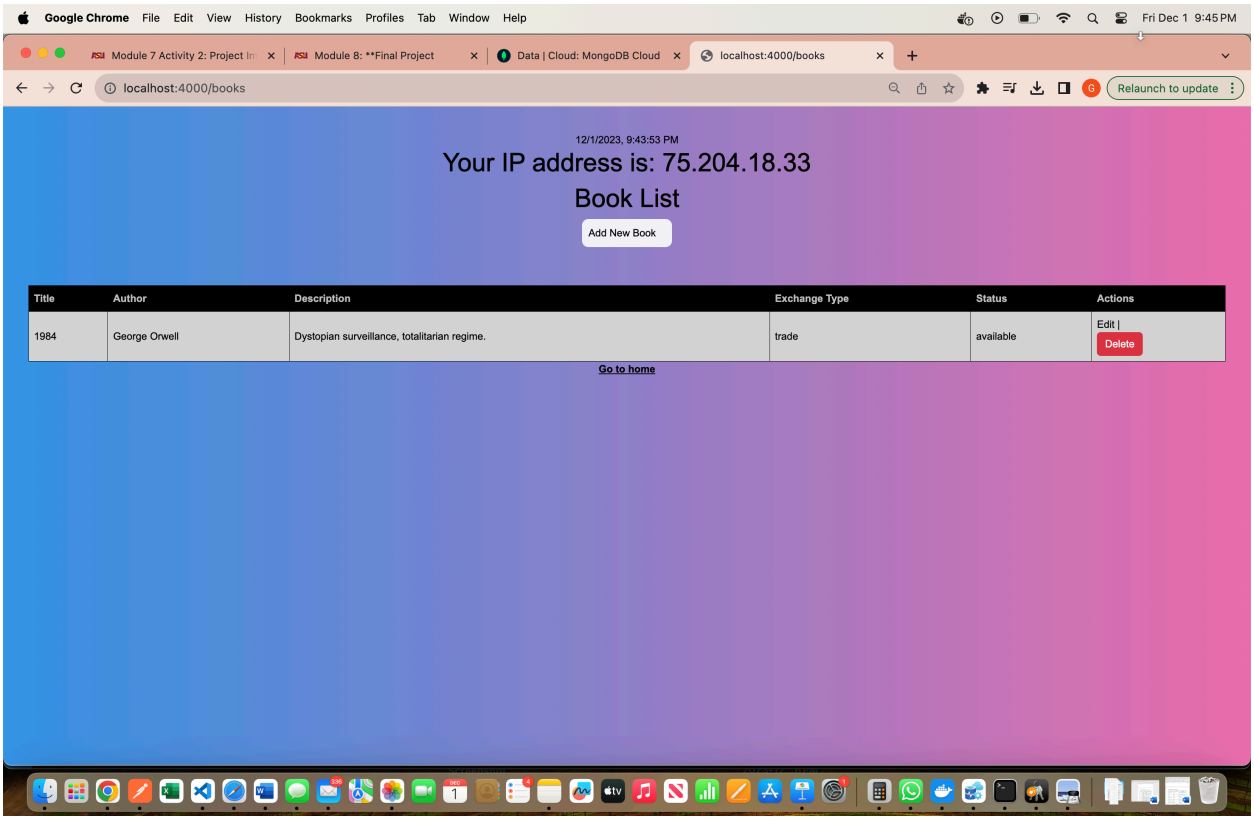
JWT Token after logging in:



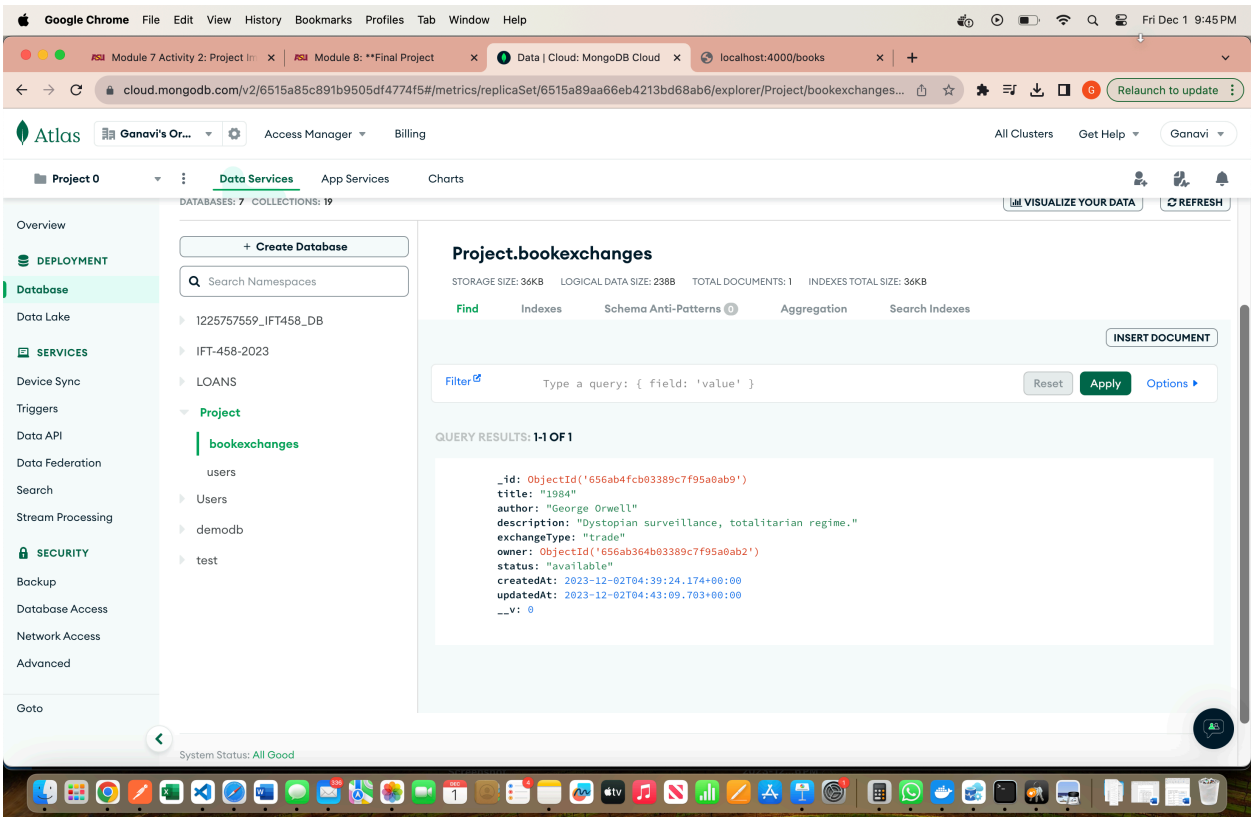
User created in MongoDb database:



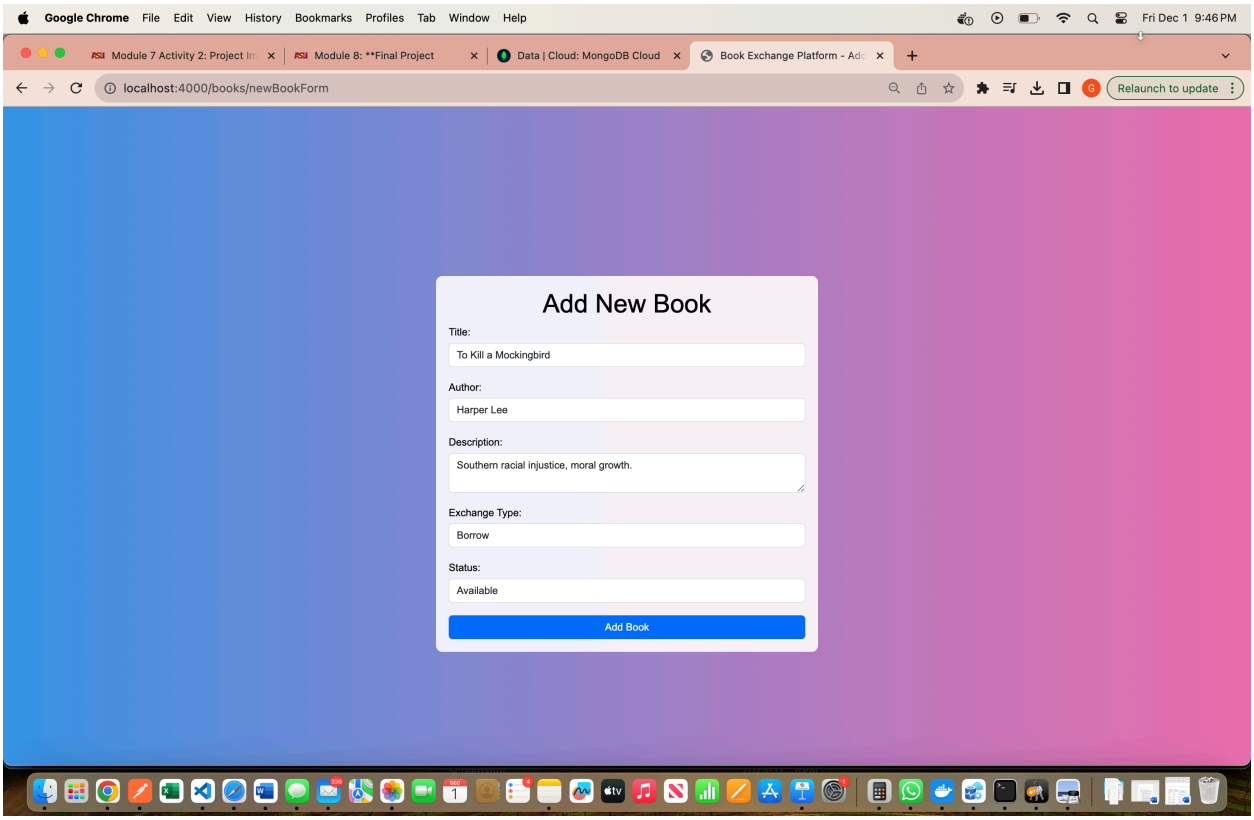
Added new book:

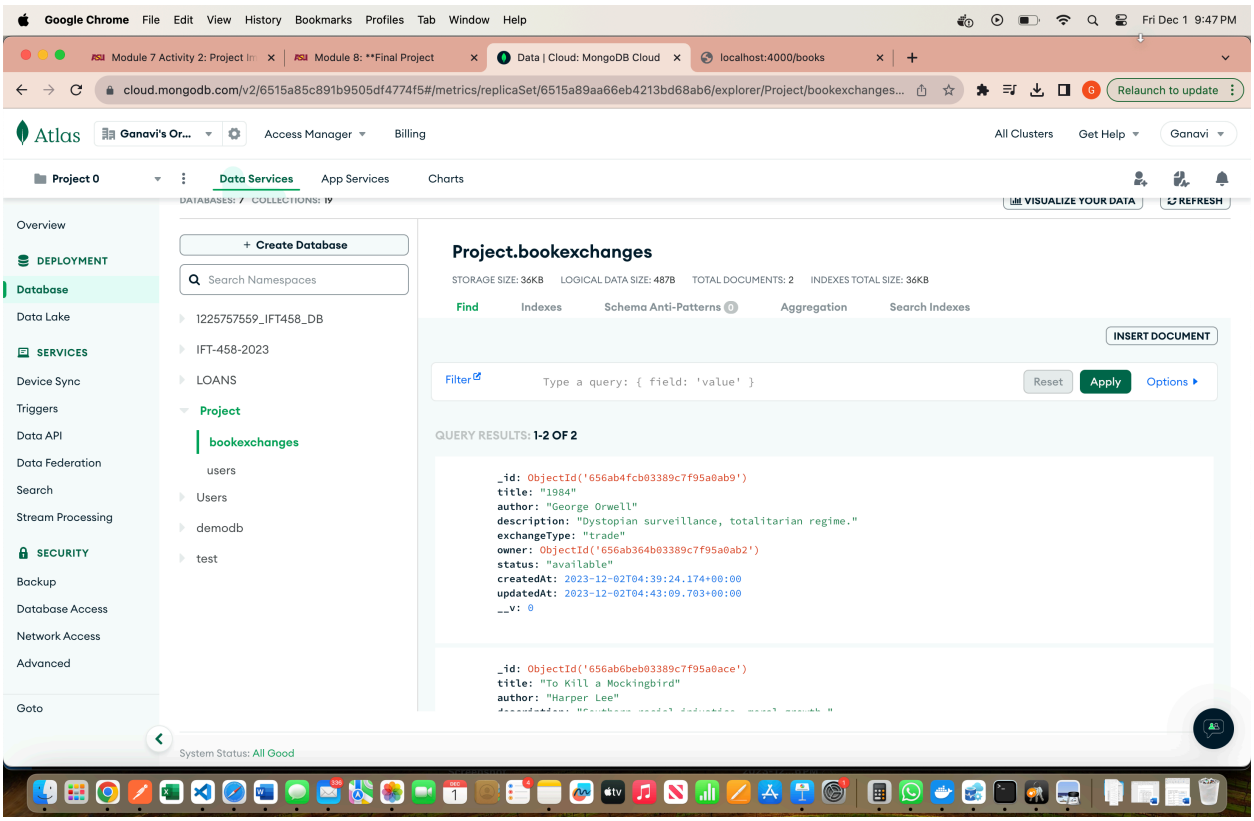


Added book in MongoDB:

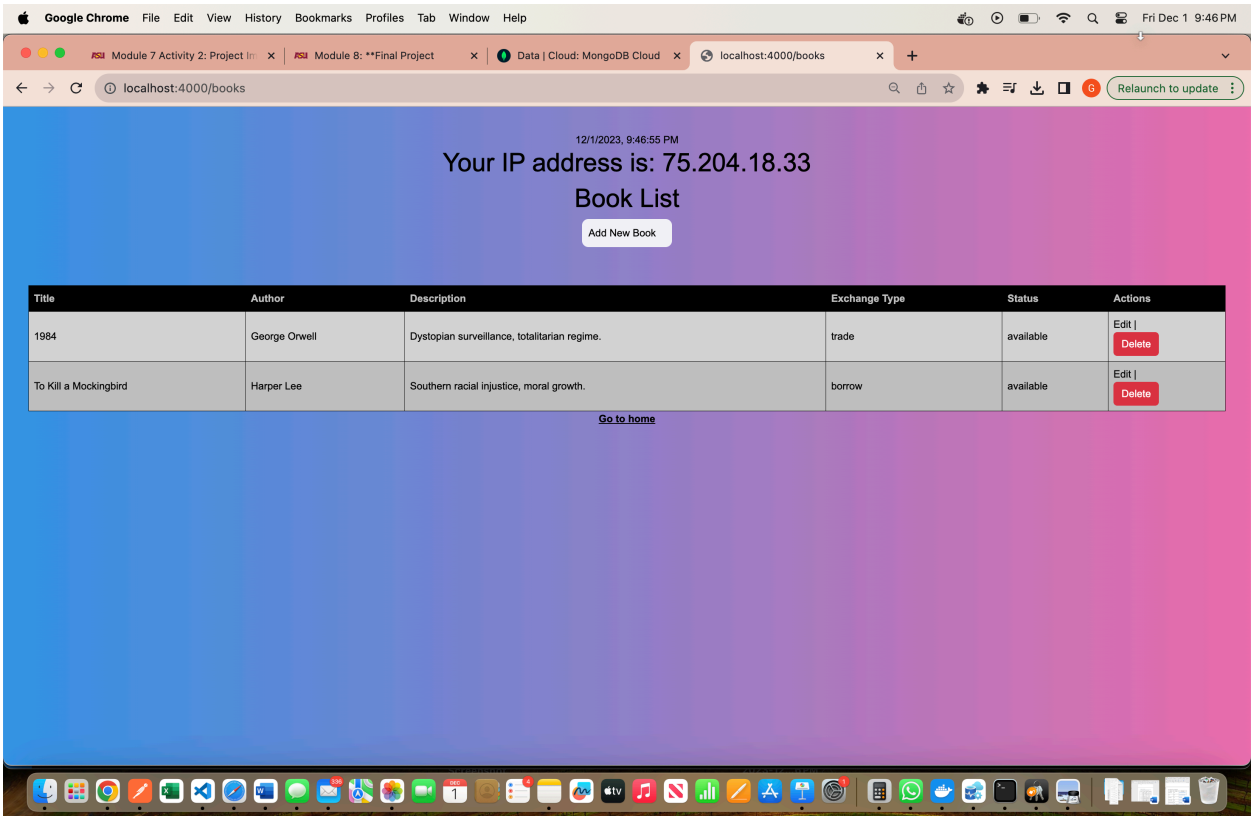


Adding 2nd book (POST):

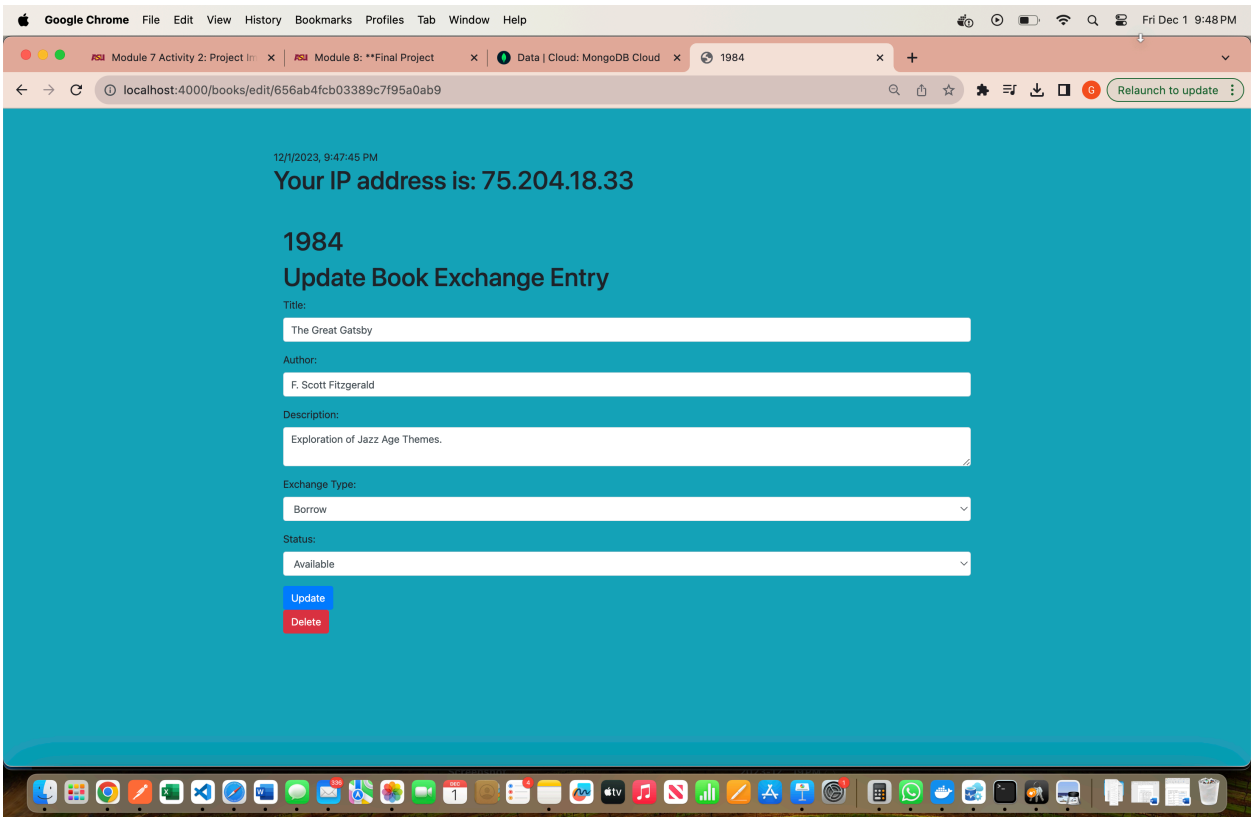




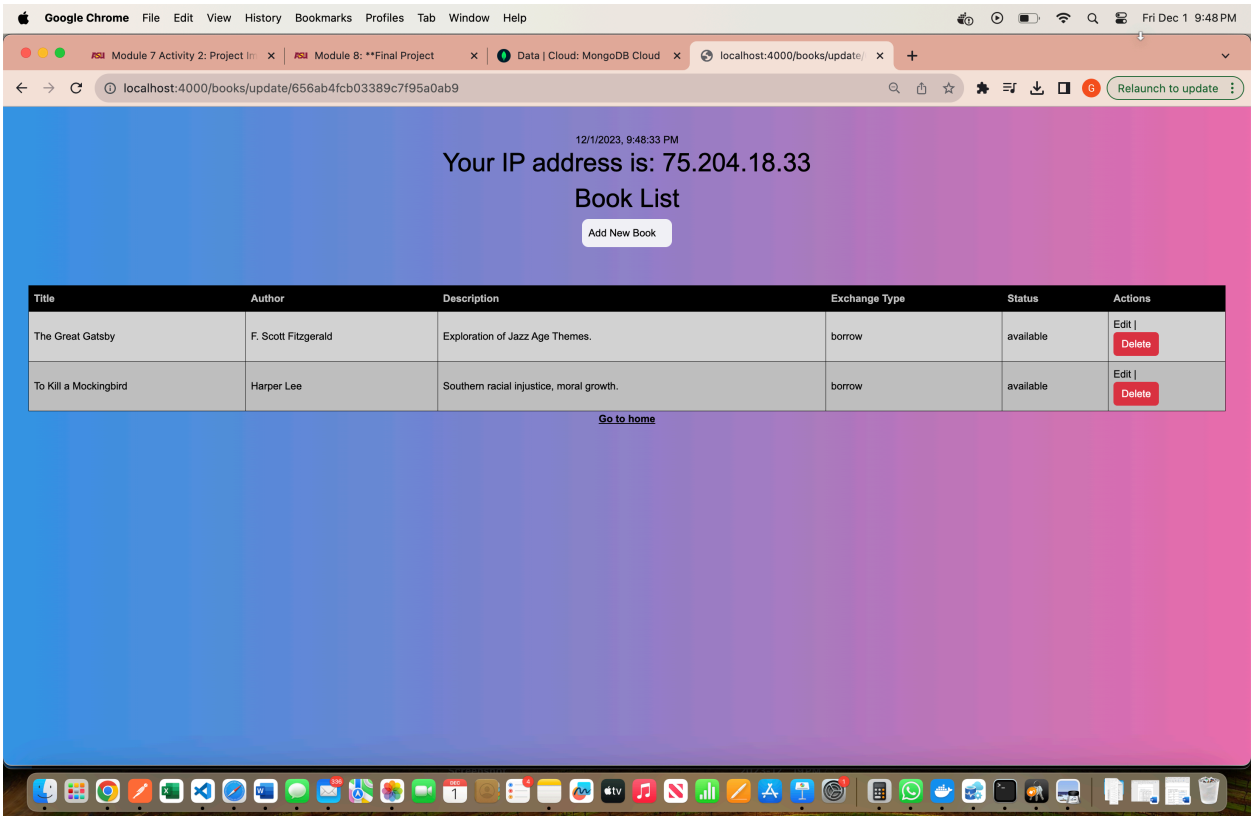
All books:

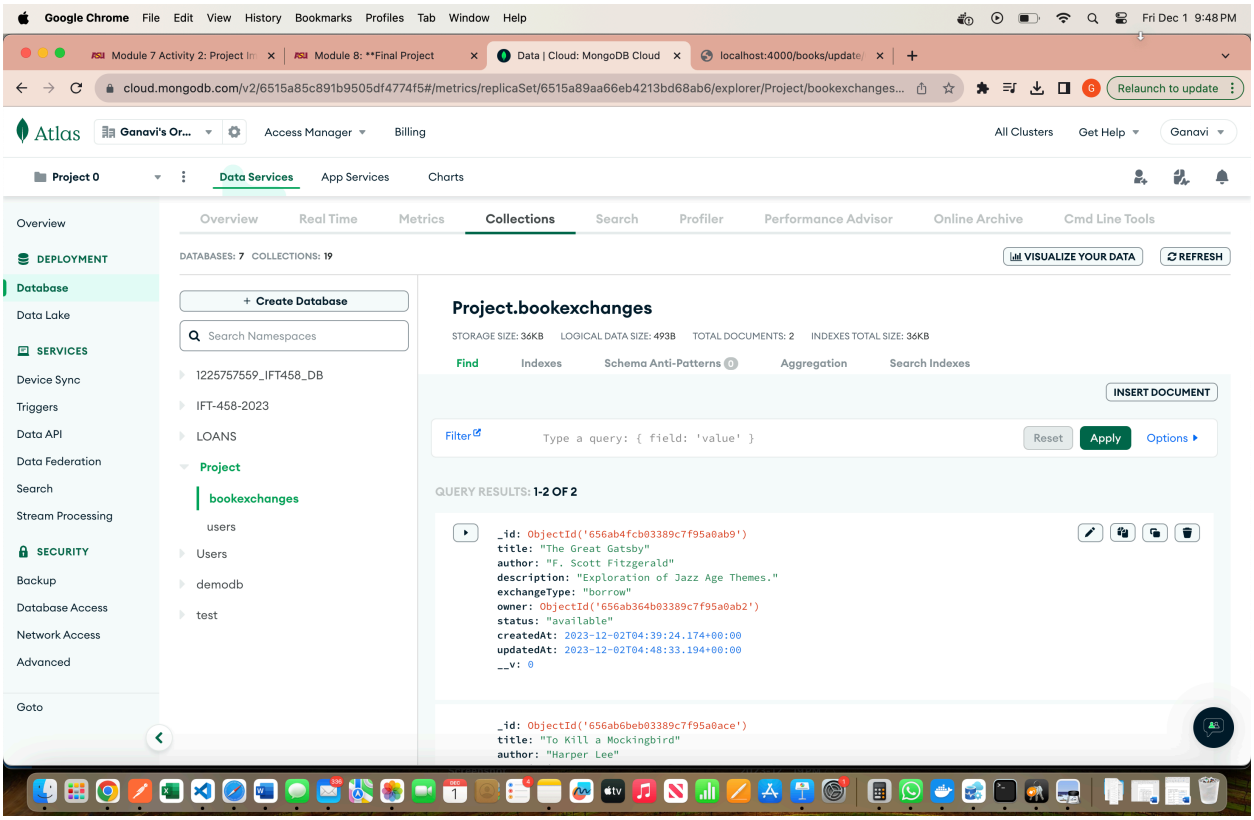


Updating the book 1 (PUT):

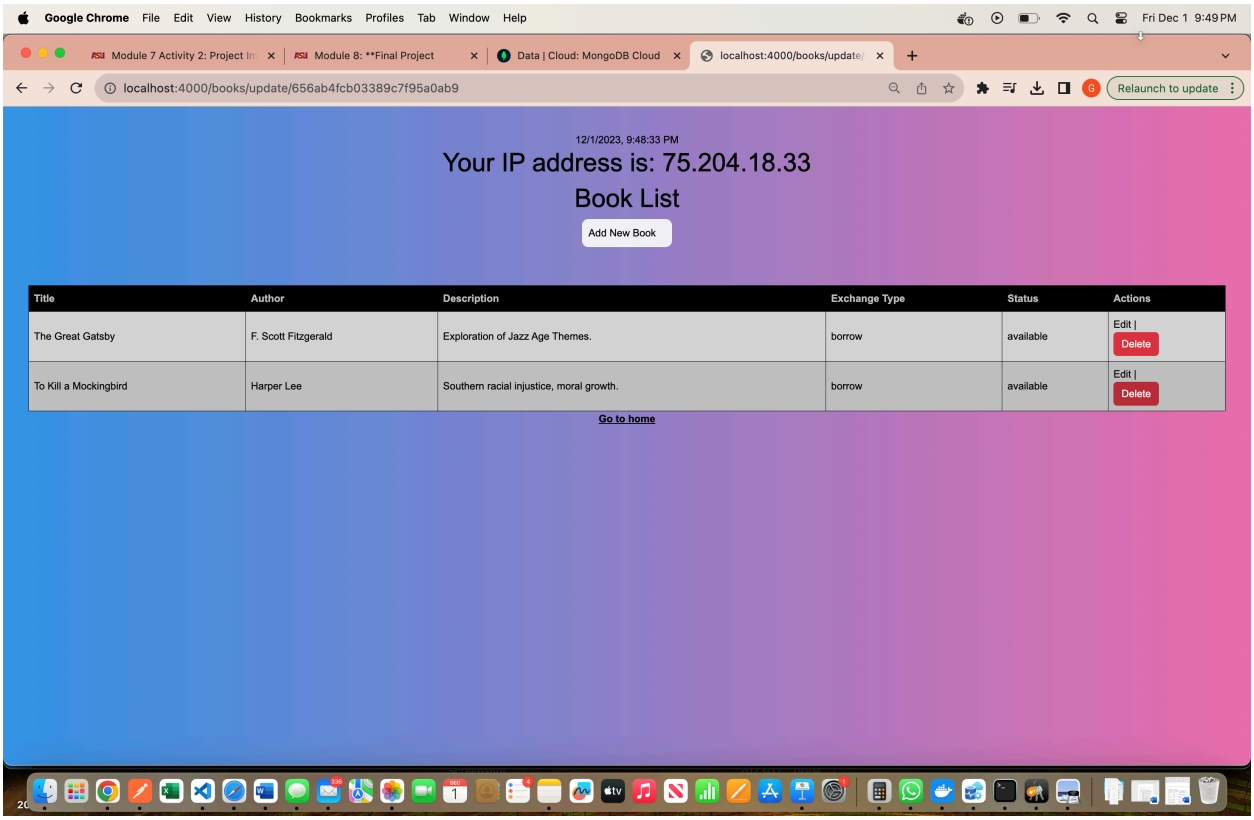


After updating the book:

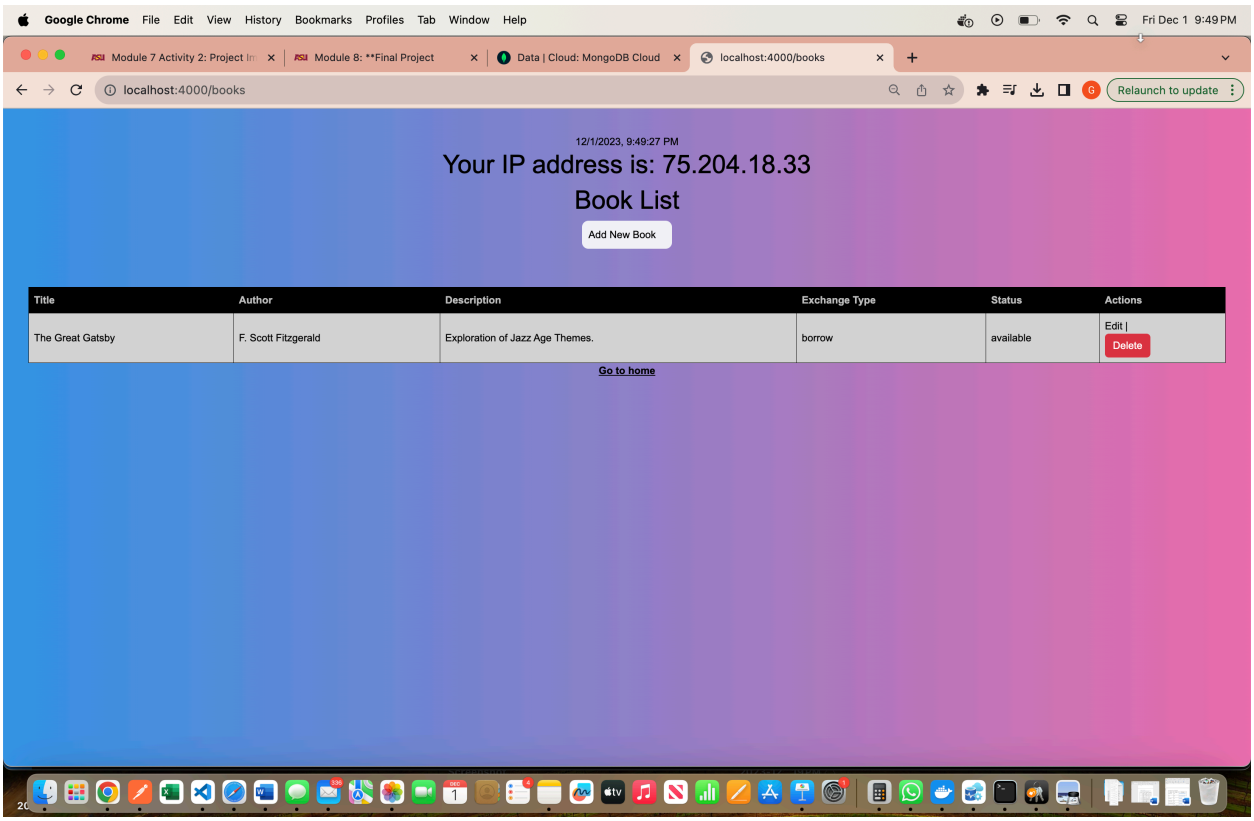


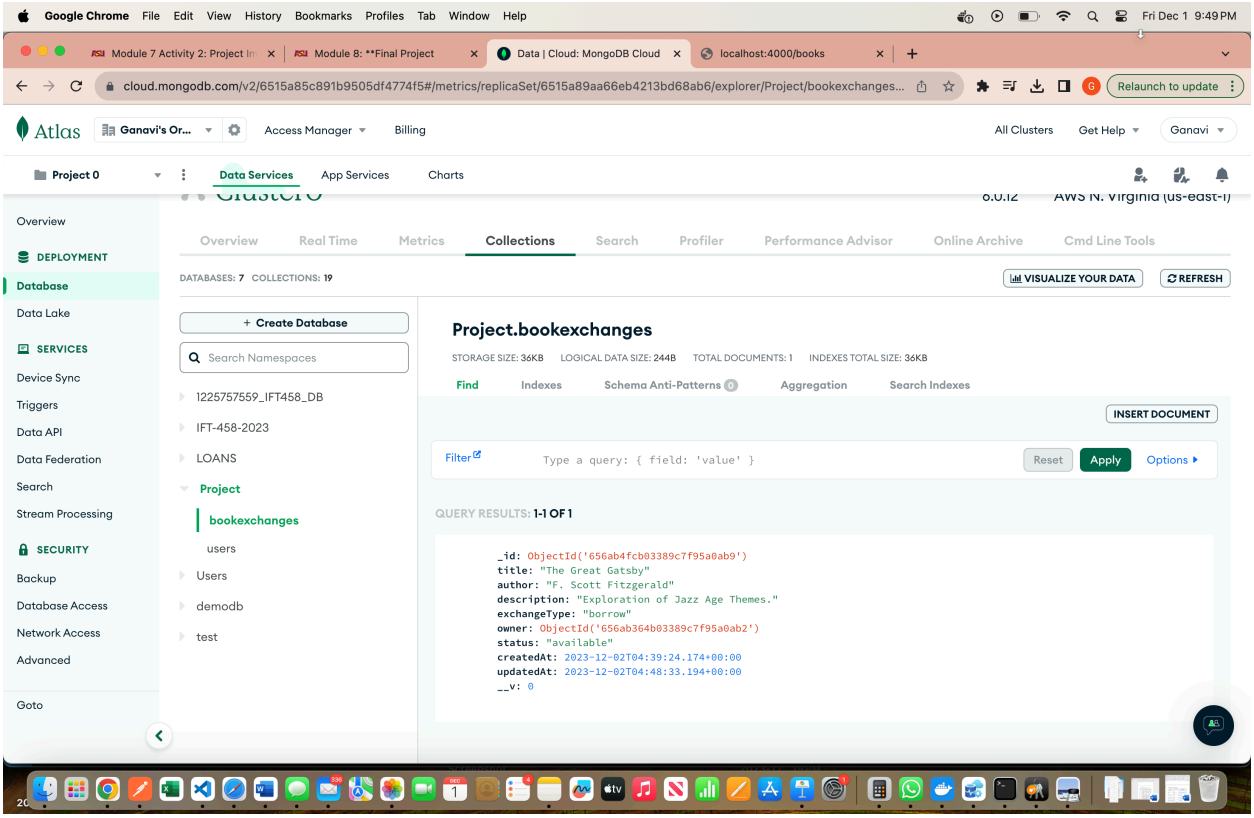


Deleting the 2nd book:



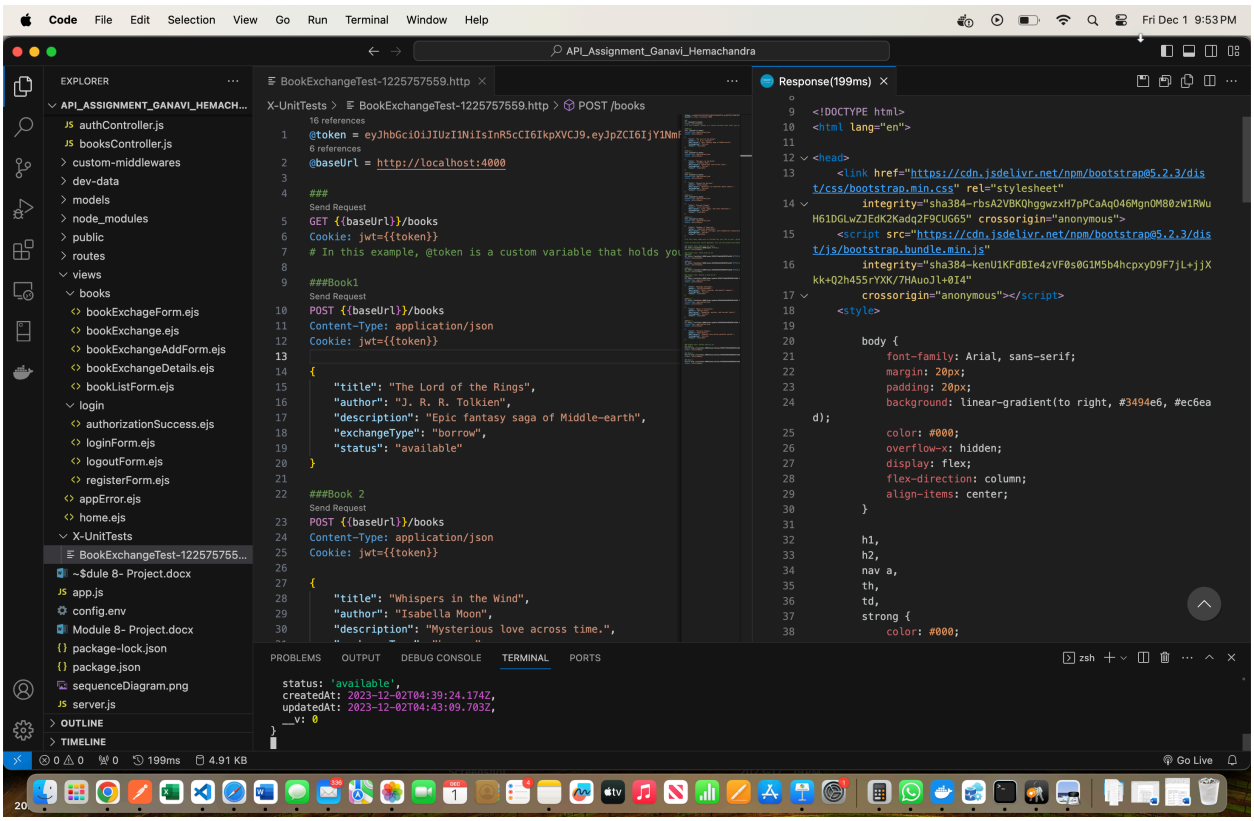
After deleting the 2nd book:

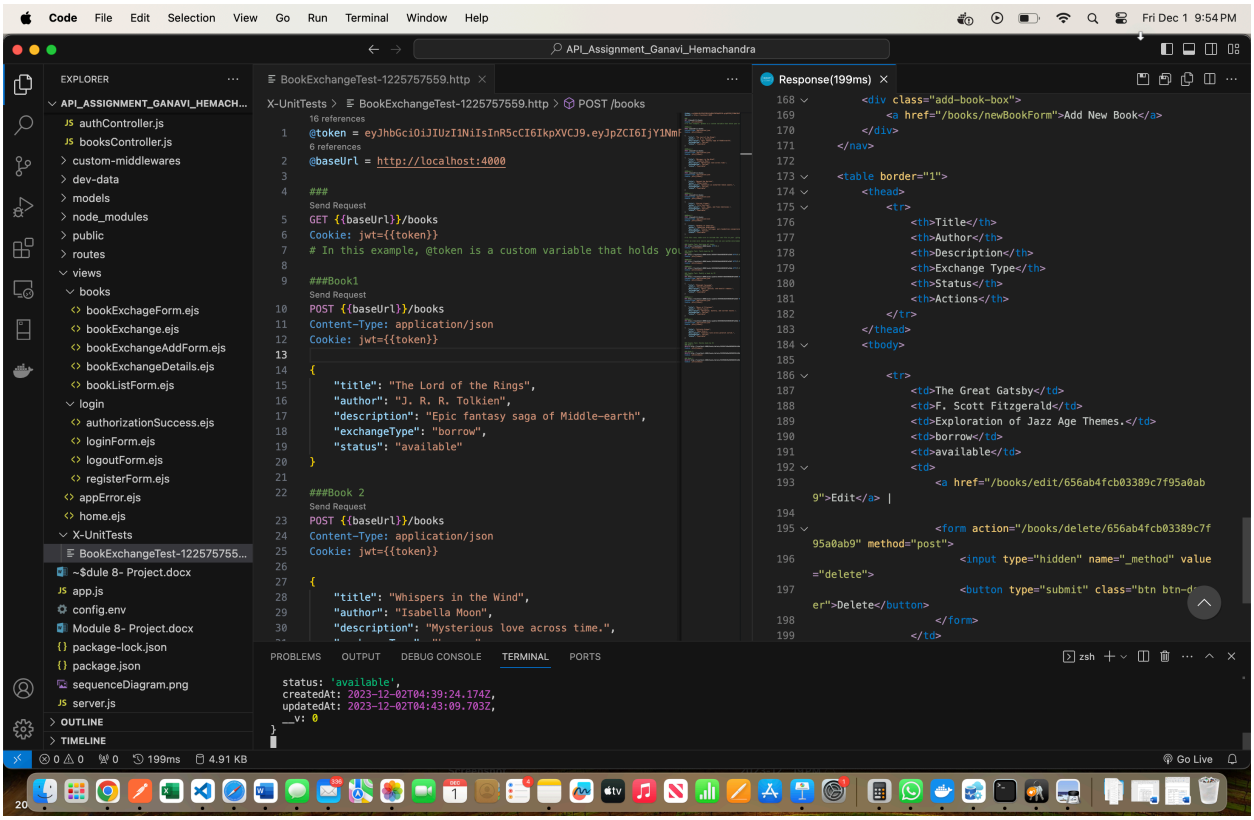




CRUD Operation (POST, GET, PUT & DELETE):

GET books:

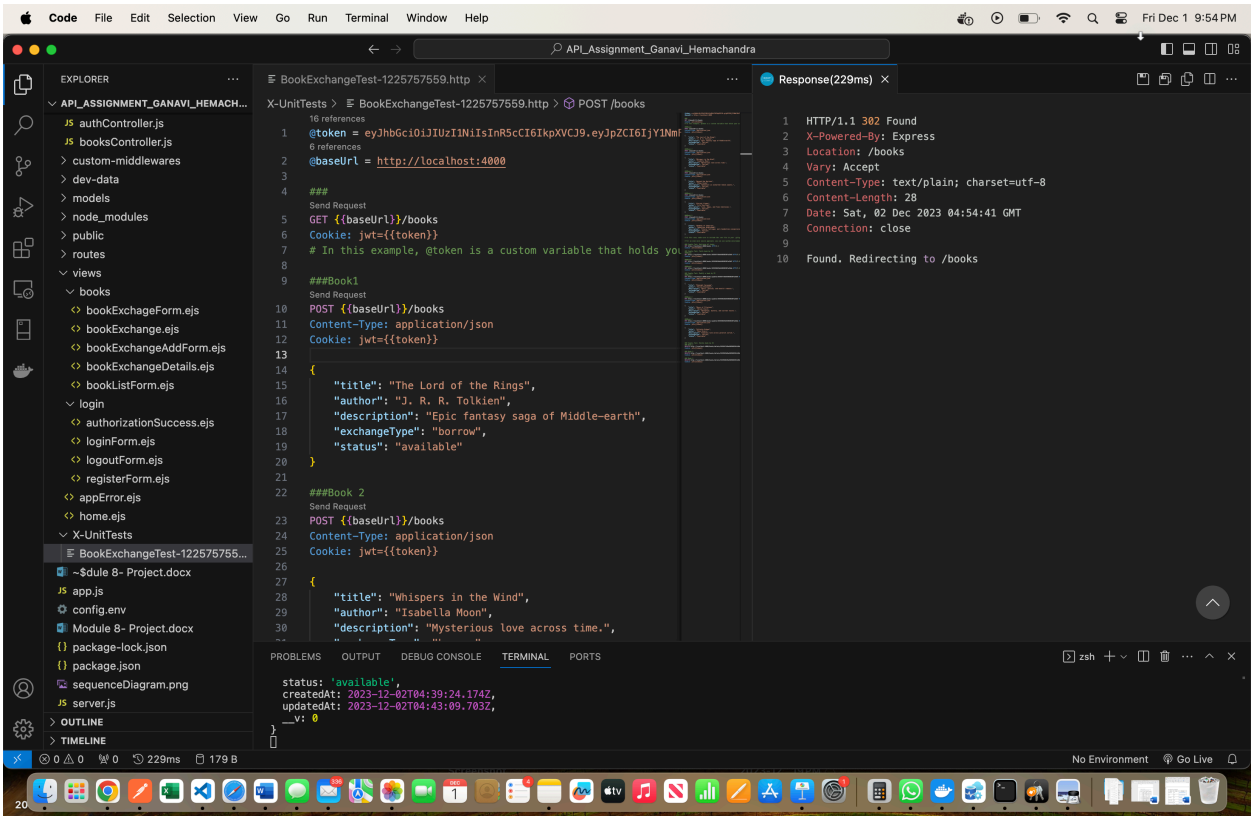


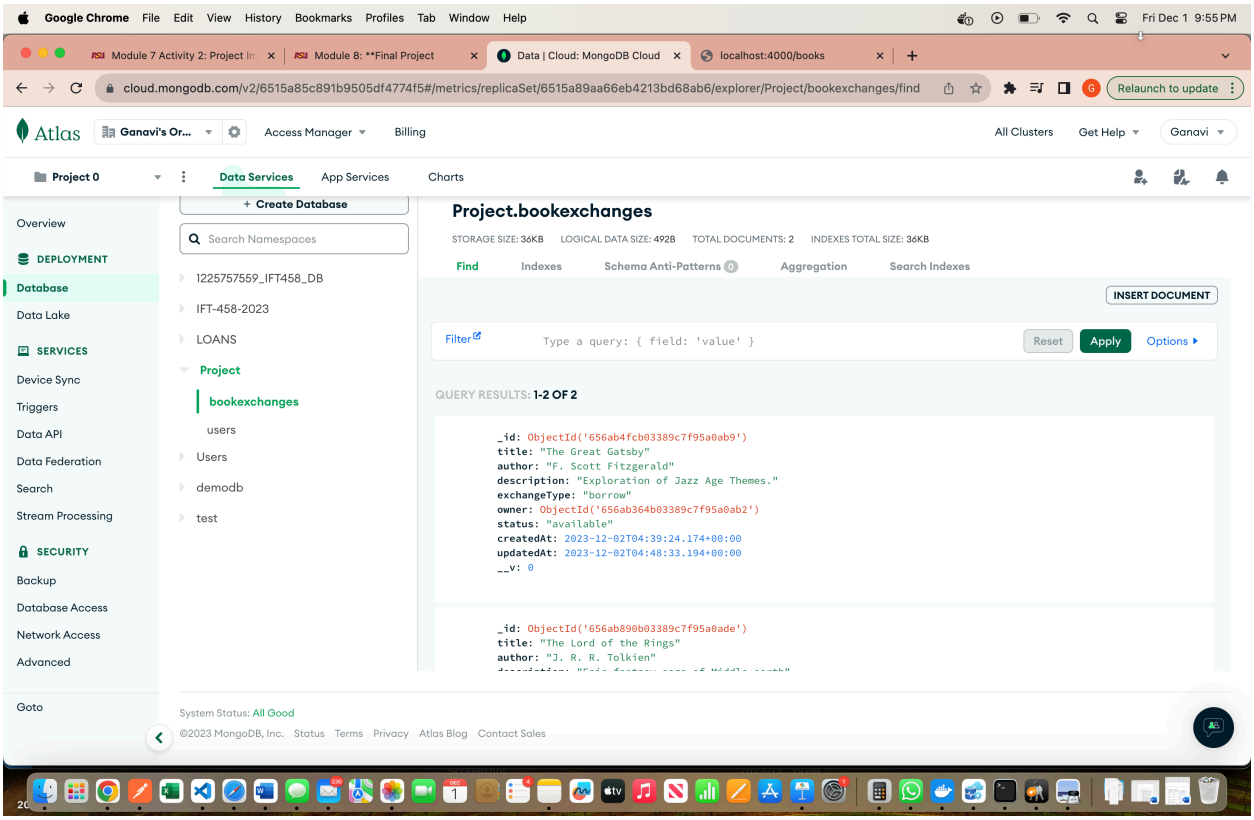

Code File Edit Selection View Go Run Terminal Window Help
Fri Dec 1 9:54 PM
API_Assignment_Ganavi_Hemachandra
EXPLORER
API_ASSIGNMENT_GANAVI_HEMACH...
authController.js
booksController.js
custom-middlewares
dev-data
models
node_modules
public
routes
views
books
bookExchageForm.ejs
bookExchange.ejs
bookExchangeAddForm.ejs
bookExchangeDetails.ejs
bookListForm.ejs
login
authorizationSuccess.ejs
loginForm.ejs
logoutForm.ejs
registerForm.ejs
appError.ejs
home.ejs
X-UnitTests
BookExchangeTest-122575755...
~$dule 8- Project.docx
app.js
config.env
Module 8- Project.docx
package-lock.json
package.json
sequenceDiagram.png
server.js
OUTLINE
TIMELINE
BookExchangeTest-1225757559.http
X-UnitTests > BookExchangeTest-1225757559.http > POST /books
16 references
@token = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1Nmf
6 references
@baseUrl = http://localhost:4000
###
Send Request
GET {{baseUrl}}/books
Cookie: jwt={{token}}
In this example, @token is a custom variable that holds yo
###Book1
Send Request
POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}
{
"title": "The Lord of the Rings",
"author": "J. R. R. Tolkien",
"description": "Epic fantasy saga of Middle-earth",
"exchangeType": "borrow",
"status": "available"
}
###Book 2
Send Request
POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}
{
"title": "Whispers in the Wind",
"author": "Isabella Moon",
"description": "Mysterious love across time.",
Response(199ms)

Add New Book

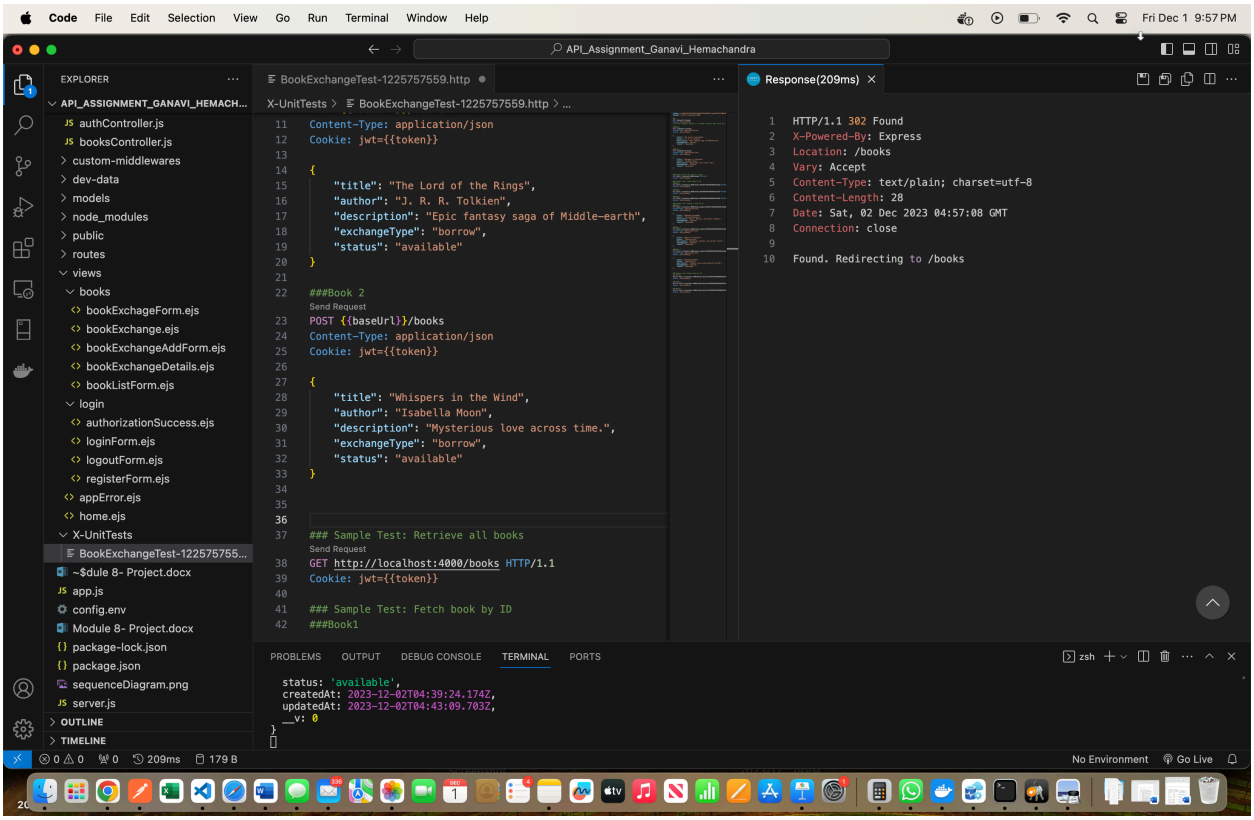
</nav>
<table border="1">
<thead>
<tr>
<th>Title</th>
<th>Author</th>
<th>Description</th>
<th>Exchange Type</th>
<th>Status</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Great Gatsby</td>
<td>F. Scott Fitzgerald</td>
<td>Exploration of Jazz Age Themes.</td>
<td>borrow</td>
<td>available</td>
<td>
Edit |
<form action="/books/delete/656ab4fcb03389c7f95a0ab9" method="post">
<input type="hidden" name="_method" value="delete">
<button type="submit" class="btn btn-danger">Delete</button>
</form>
</td>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
zsh
status: 'available',
createdAt: 2023-12-02T04:39:24.174Z,
updatedAt: 2023-12-02T04:43:09.703Z,
__v: 0
}
0 0 0 199ms 4.91 KB
Go Live

POST book 1:

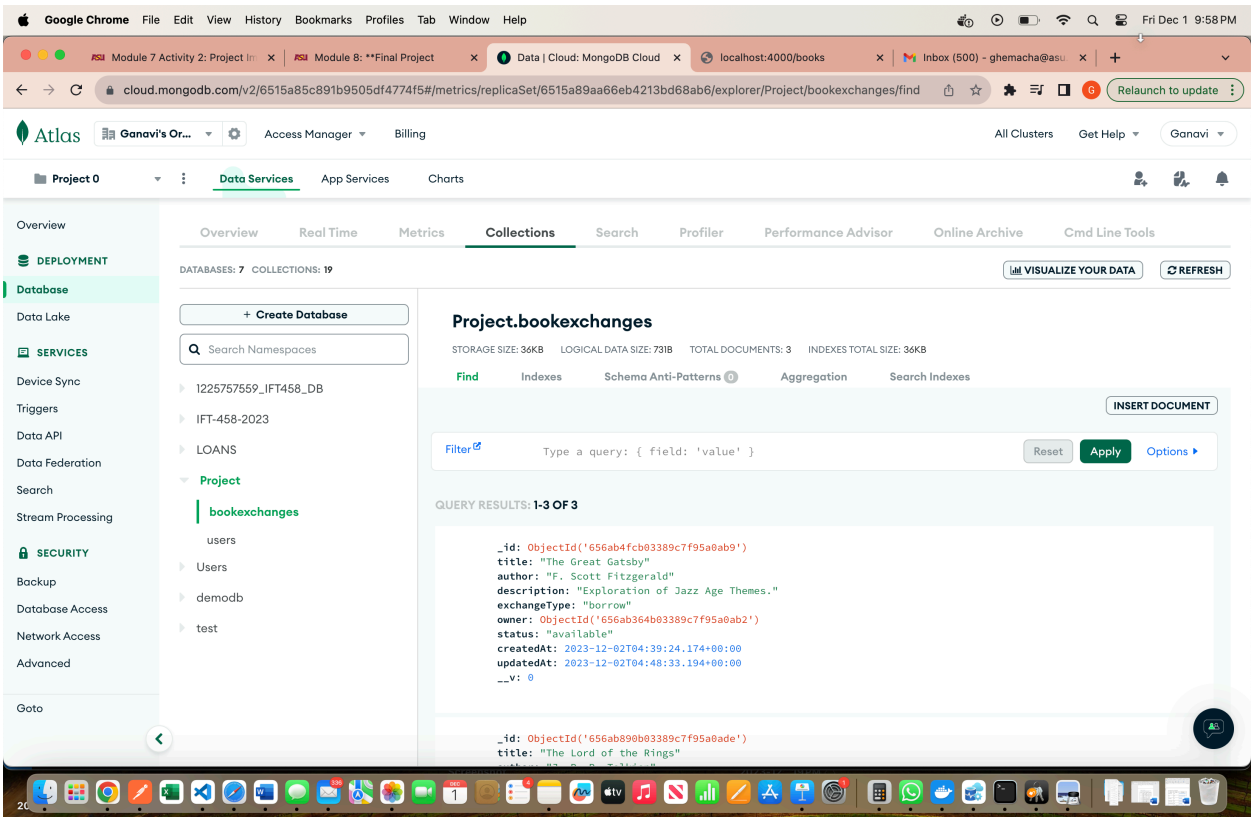

Code File Edit Selection View Go Run Terminal Window Help
Fri Dec 1 9:54 PM
API_Assignment_Ganavi_Hemachandra
EXPLORER
API_ASSIGNMENT_GANAVI_HEMACH...
authController.js
booksController.js
custom-middlewares
dev-data
models
node_modules
public
routes
views
books
bookExchageForm.ejs
bookExchange.ejs
bookExchangeAddForm.ejs
bookExchangeDetails.ejs
bookListForm.ejs
login
authorizationSuccess.ejs
loginForm.ejs
logoutForm.ejs
registerForm.ejs
appError.ejs
home.ejs
X-UnitTests
BookExchangeTest-122575755...
~$dule 8- Project.docx
app.js
config.env
Module 8- Project.docx
package-lock.json
package.json
sequenceDiagram.png
server.js
OUTLINE
TIMELINE
BookExchangeTest-1225757559.http
X-UnitTests > BookExchangeTest-1225757559.http > POST /books
16 references
@token = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1Nmf
6 references
@baseUrl = http://localhost:4000
###
Send Request
GET {{baseUrl}}/books
Cookie: jwt={{token}}
In this example, @token is a custom variable that holds yo
###Book1
Send Request
POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}
{
"title": "The Lord of the Rings",
"author": "J. R. R. Tolkien",
"description": "Epic fantasy saga of Middle-earth",
"exchangeType": "borrow",
"status": "available"
}
###Book 2
Send Request
POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}
{
"title": "Whispers in the Wind",
"author": "Isabella Moon",
"description": "Mysterious love across time.",
Response(229ms)
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /books
Vary: Accept
Content-Type: text/plain; charset=utf-8
Content-Length: 28
Date: Sat, 02 Dec 2023 04:54:41 GMT
Connection: close
Found. Redirecting to /books
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
zsh
status: 'available',
createdAt: 2023-12-02T04:39:24.174Z,
updatedAt: 2023-12-02T04:43:09.703Z,
__v: 0
}
0 0 0 229ms 179 B
No Environment
Go Live

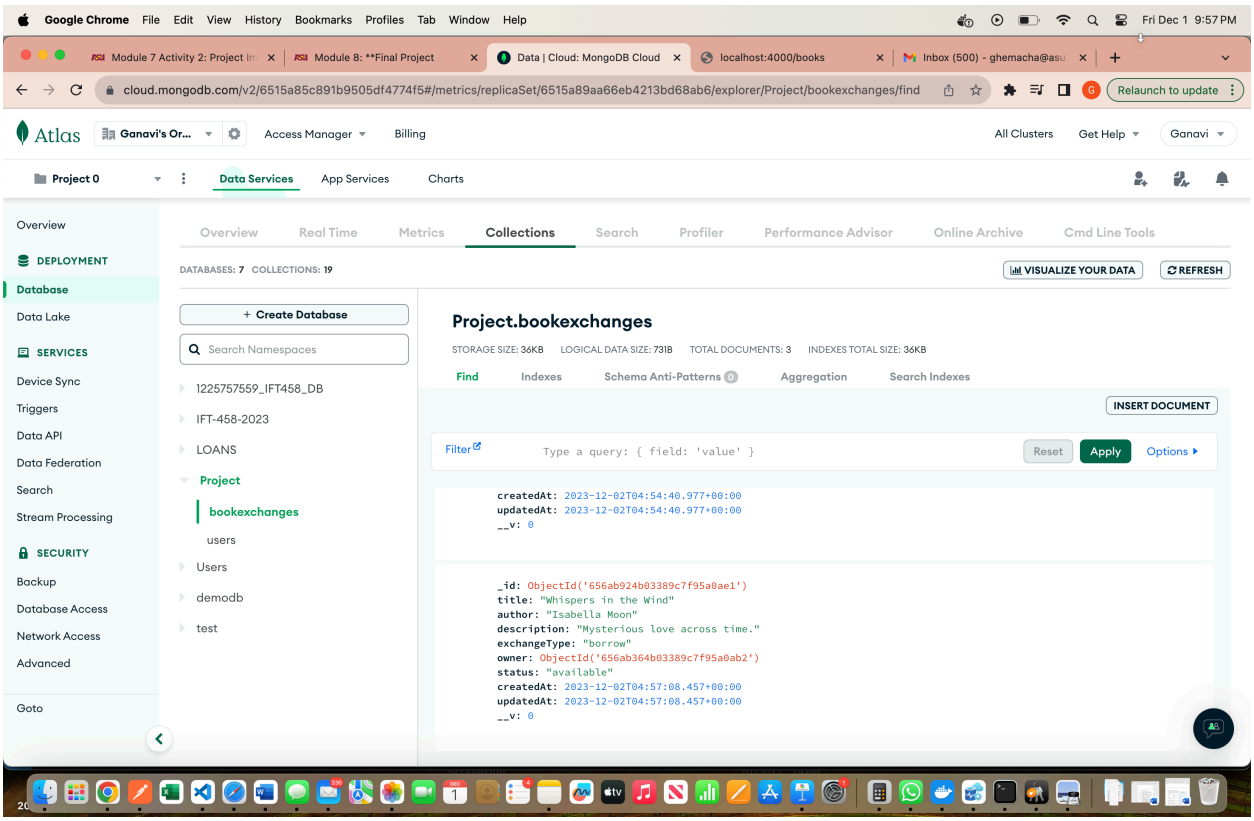

Project.bookexchanges
STORAGE SIZE: 36KB
LOGICAL DATA SIZE: 492B
TOTAL DOCUMENTS: 2
INDEXES TOTAL SIZE: 36KB
Find
Indexes
Schema Anti-Patterns
Aggregation
Search Indexes
INSERT DOCUMENT
QUERY RESULTS: 1-2 OF 2
_id: ObjectId('656ab4fcb03389c7f95a0ab9')
title: "The Great Gatsby"
author: "F. Scott Fitzgerald"
description: "Exploration of Jazz Age Themes."
exchangeType: "borrow"
owner: ObjectId('656ab364b03389c7f95a0ab2')
status: "available"
createdAt: 2023-12-02T04:39:24.174+00:00
updatedAt: 2023-12-02T04:48:33.194+00:00
__v: 0
_id: ObjectId('656ab890b03389c7f95a0ade')
title: "The Lord of the Rings"
author: "J. R. R. Tolkien"

POST Book 2:

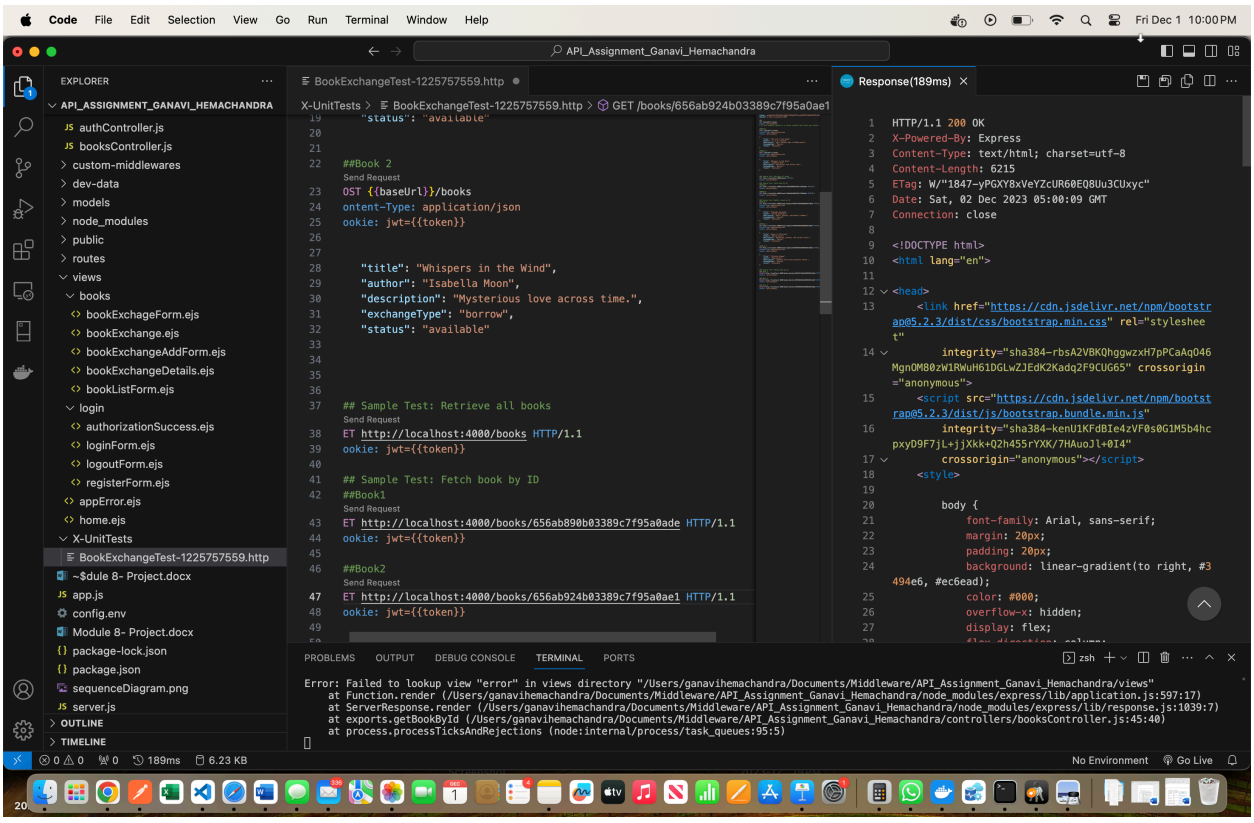

BookExchangeTest-1225757559.http
Response(209ms)
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /books
Vary: Accept
Content-Type: text/plain; charset=utf-8
Content-Length: 28
Date: Sat, 02 Dec 2023 04:57:08 GMT
Connection: close
Found. Redirecting to /books
"title": "The Lord of the Rings",
"author": "J. R. R. Tolkien",
"description": "Epic fantasy saga of Middle-earth",
"exchangeType": "borrow",
"status": "available"
###Book 2
POST {{baseUrl}}/books
"title": "Whispers in the Wind",
"author": "Isabella Moon",
"description": "Mysterious love across time.",
"exchangeType": "borrow",
"status": "available"
Sample Test: Retrieve all books
GET http://localhost:4000/books HTTP/1.1
Sample Test: Fetch book by ID
###Book1
status: 'available',
createdAt: 2023-12-02T04:39:24.174Z,
updatedAt: 2023-12-02T04:43:09.703Z,
__v: 0

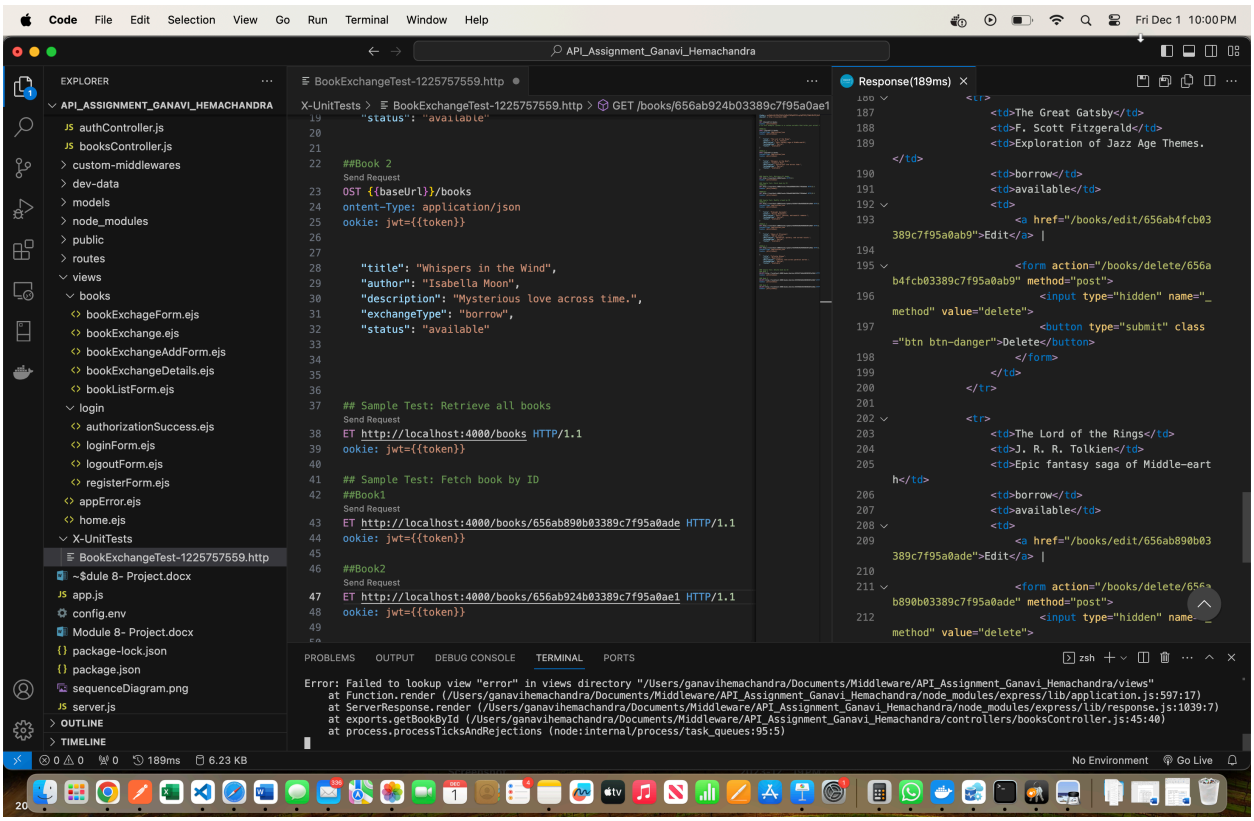
Posted book in MongoDB:

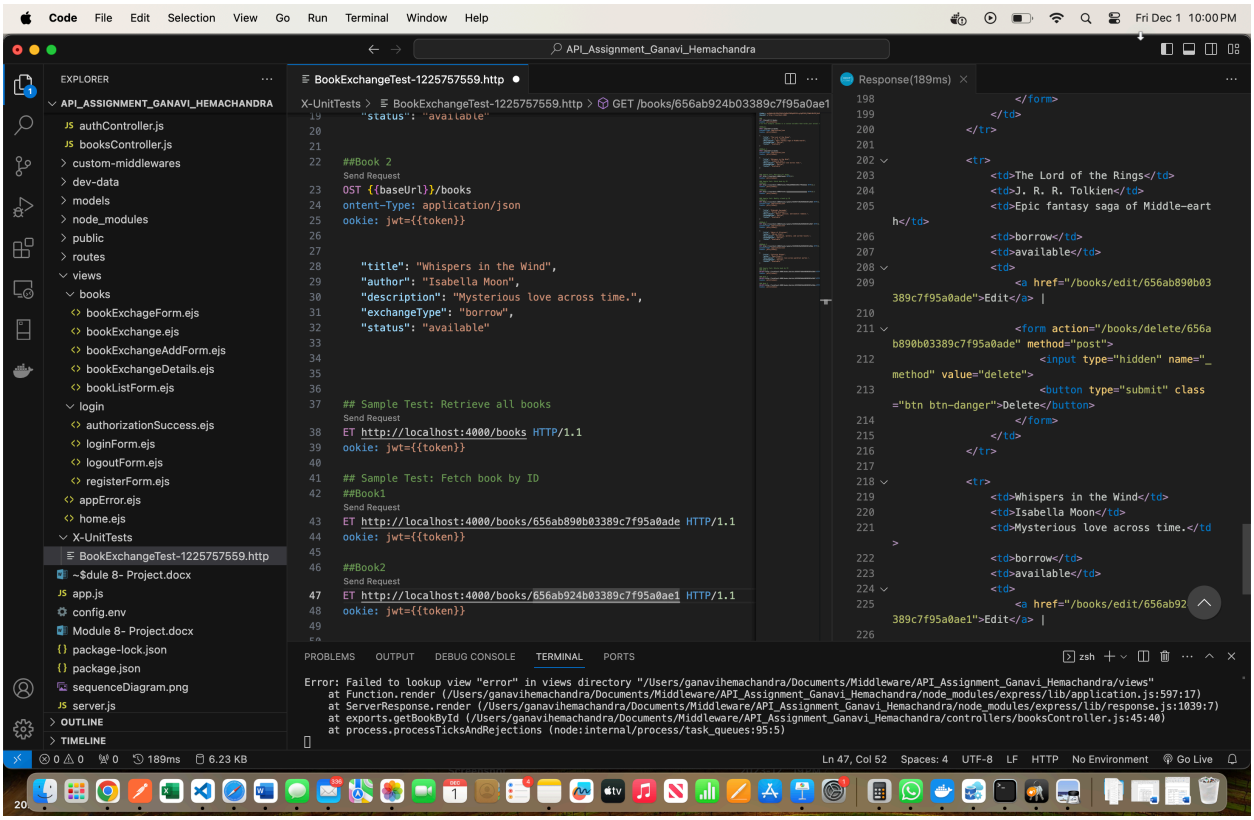




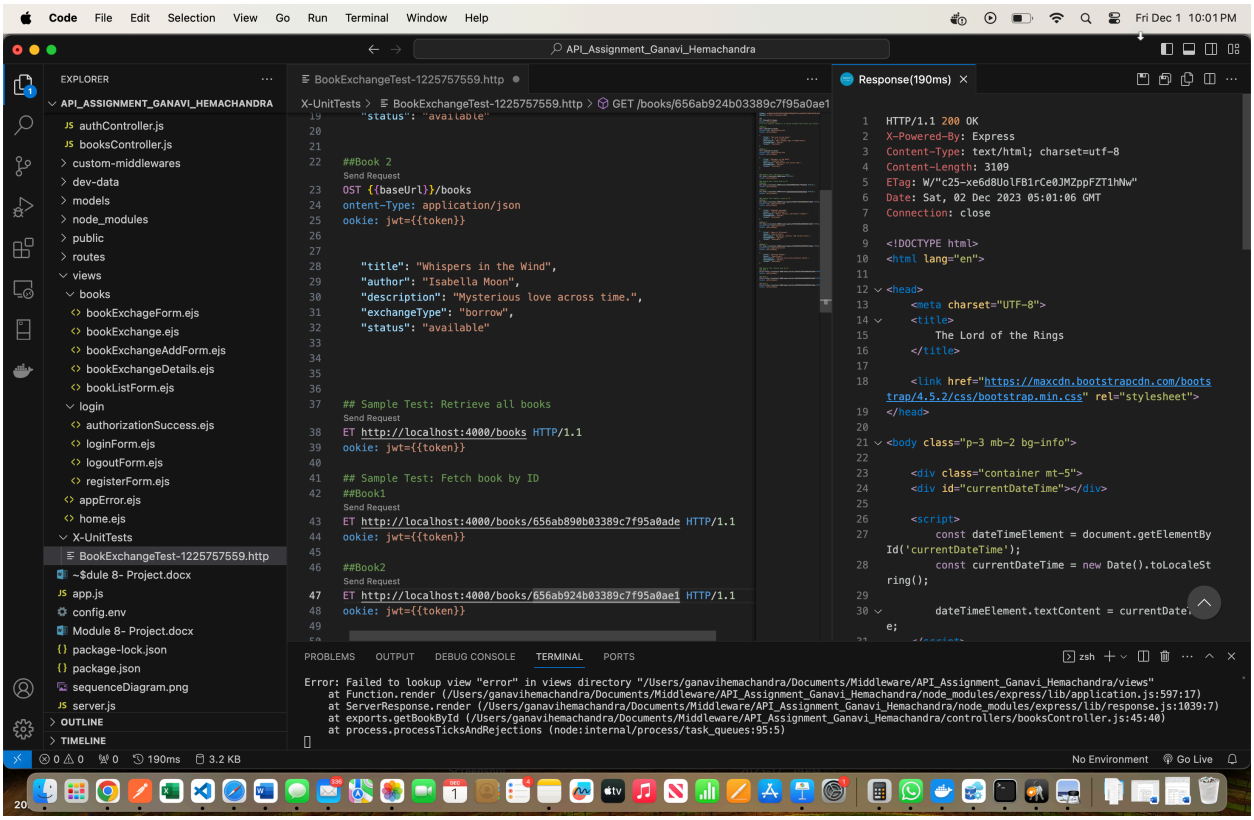
GET all books:

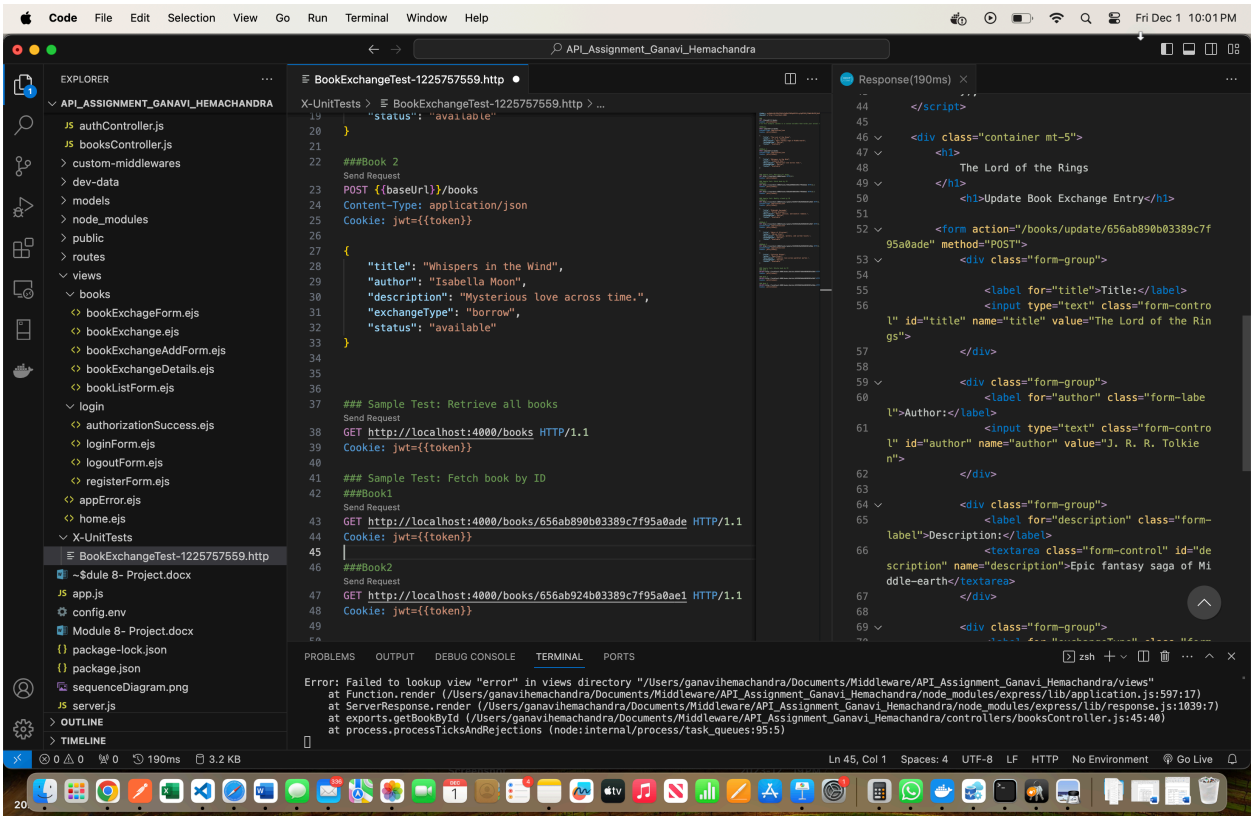




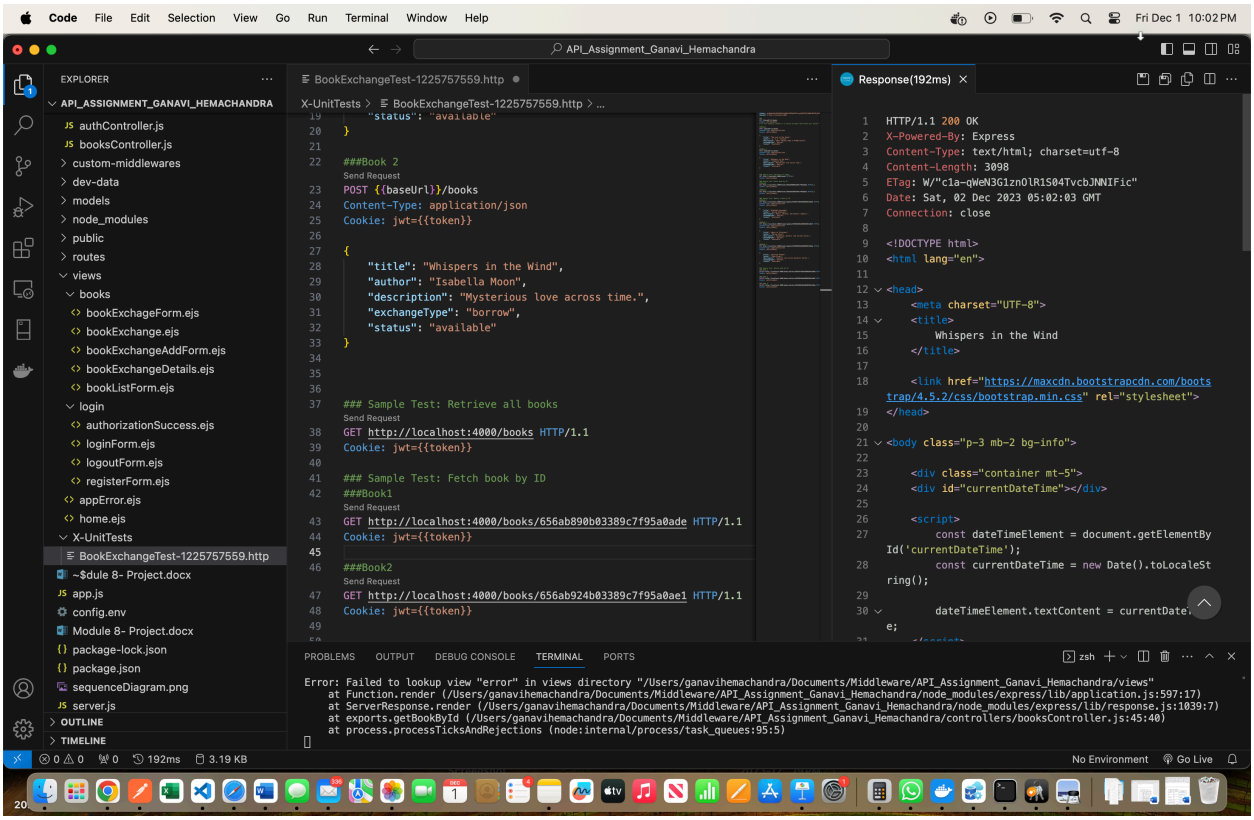


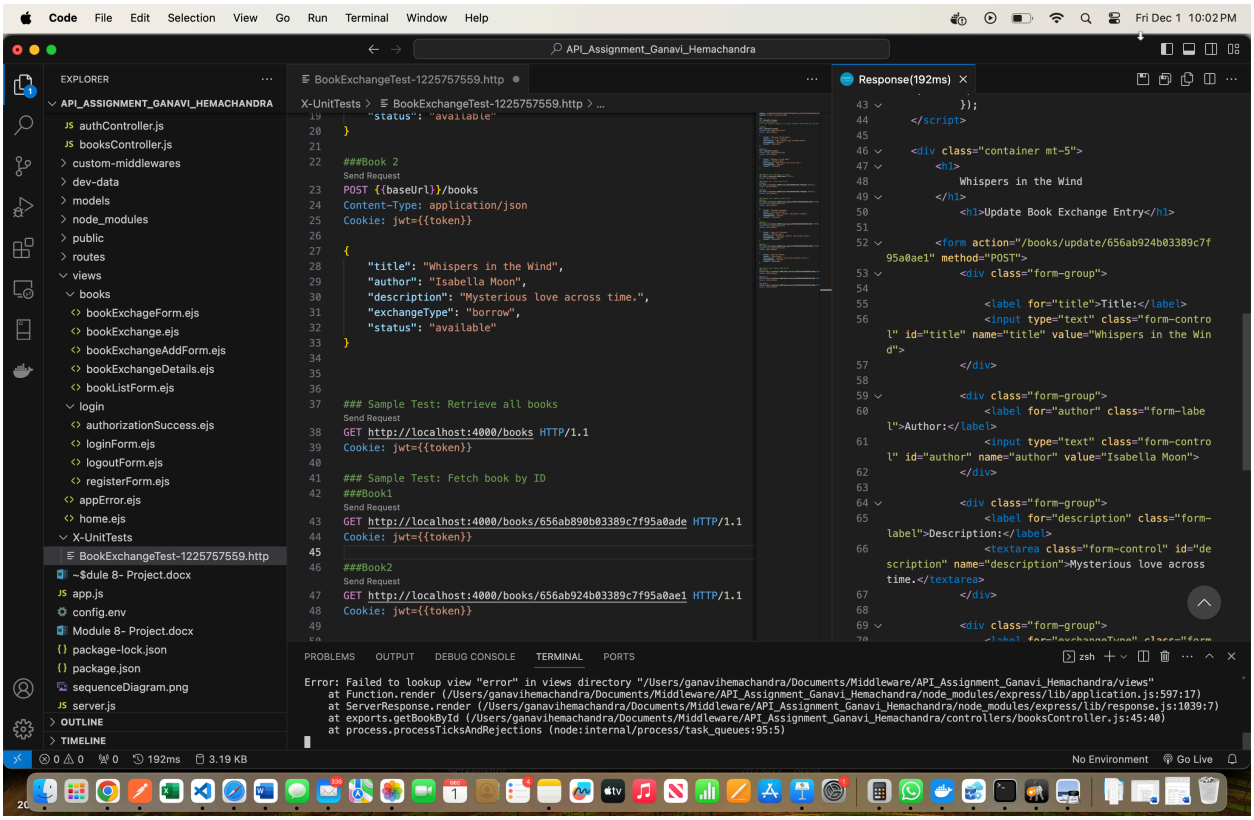
GET Book 1 by ID:



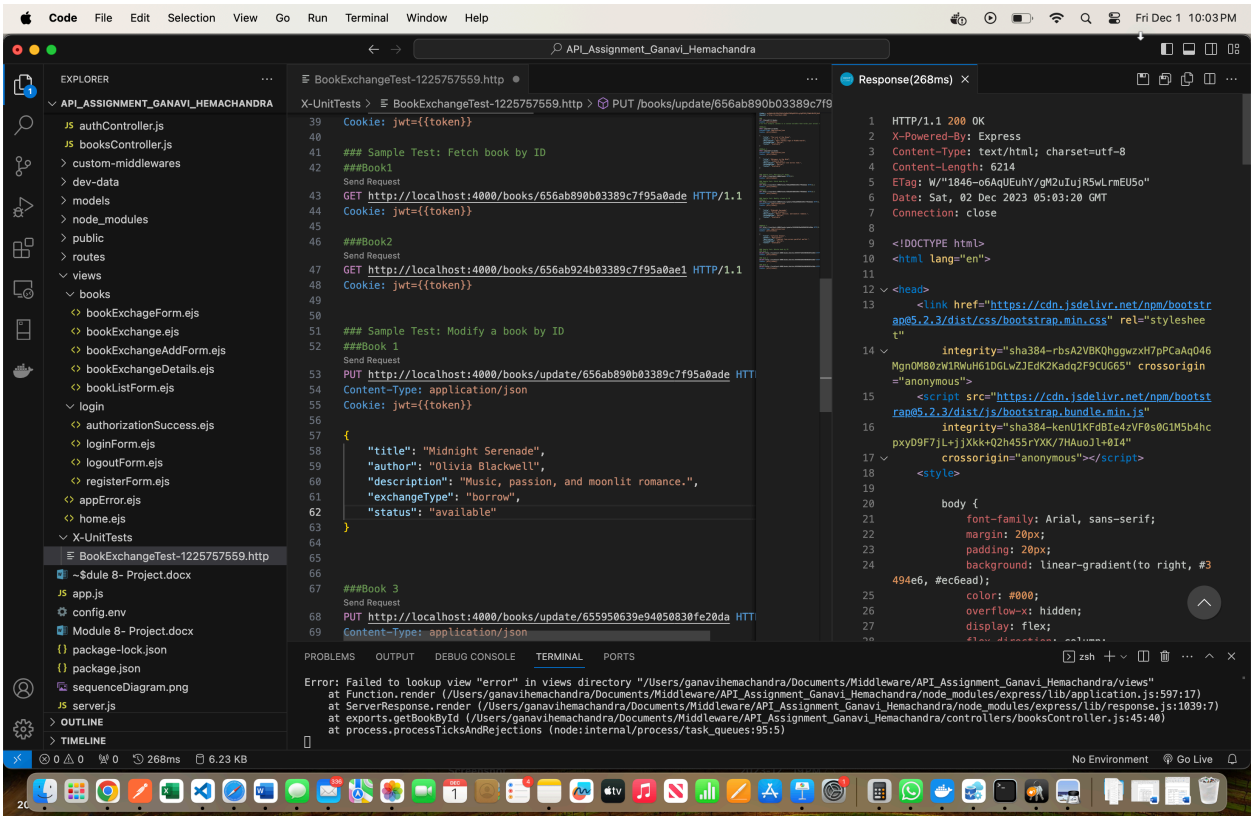


GET book 2 by ID:

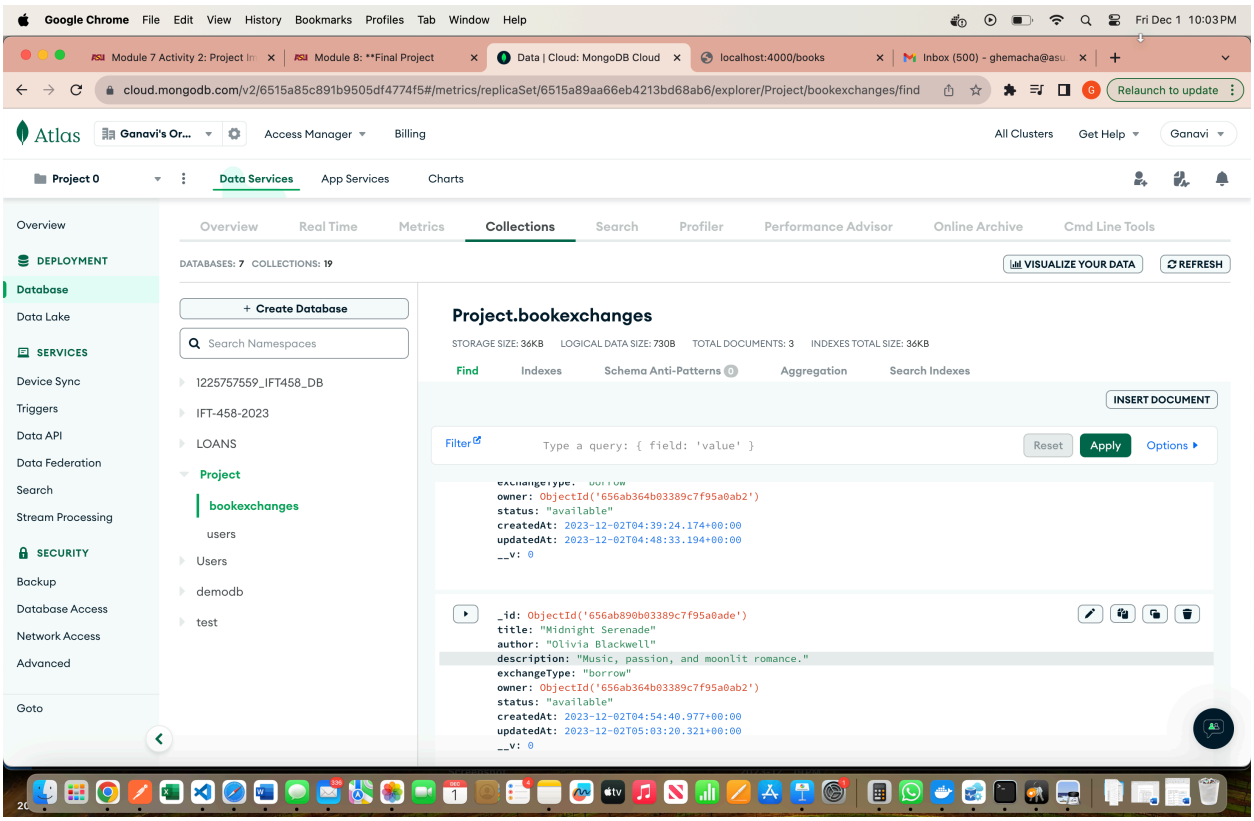




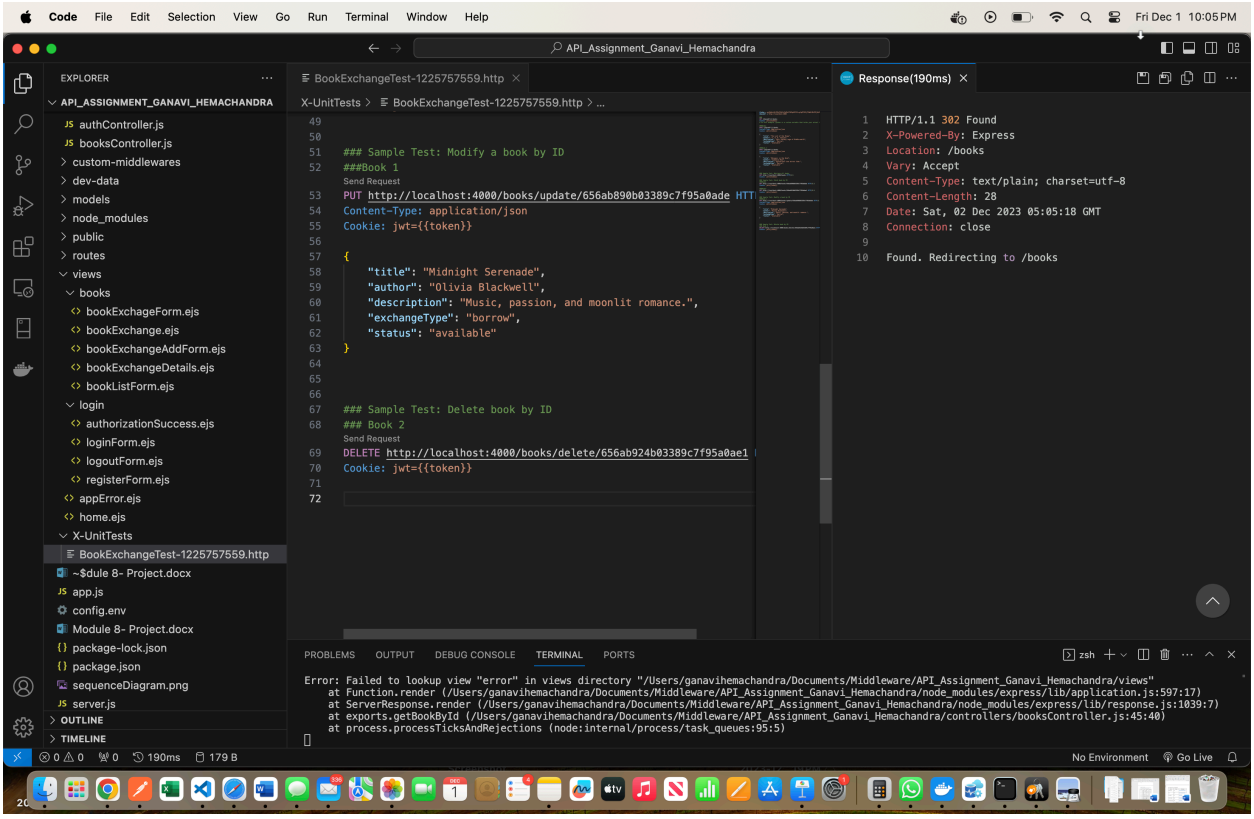
PUT book 1 by ID:



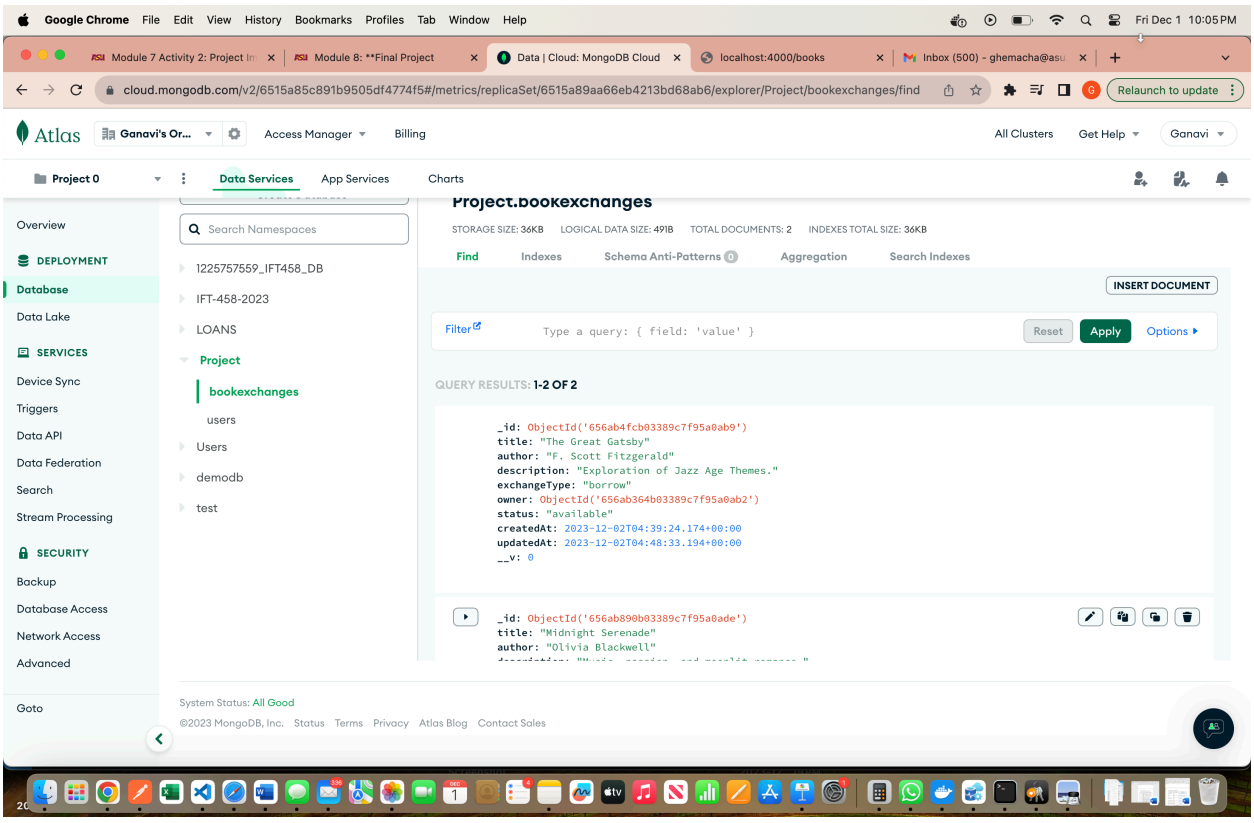
It is in 2nd place because, before posting book 1 and book 2, there already existed one book.



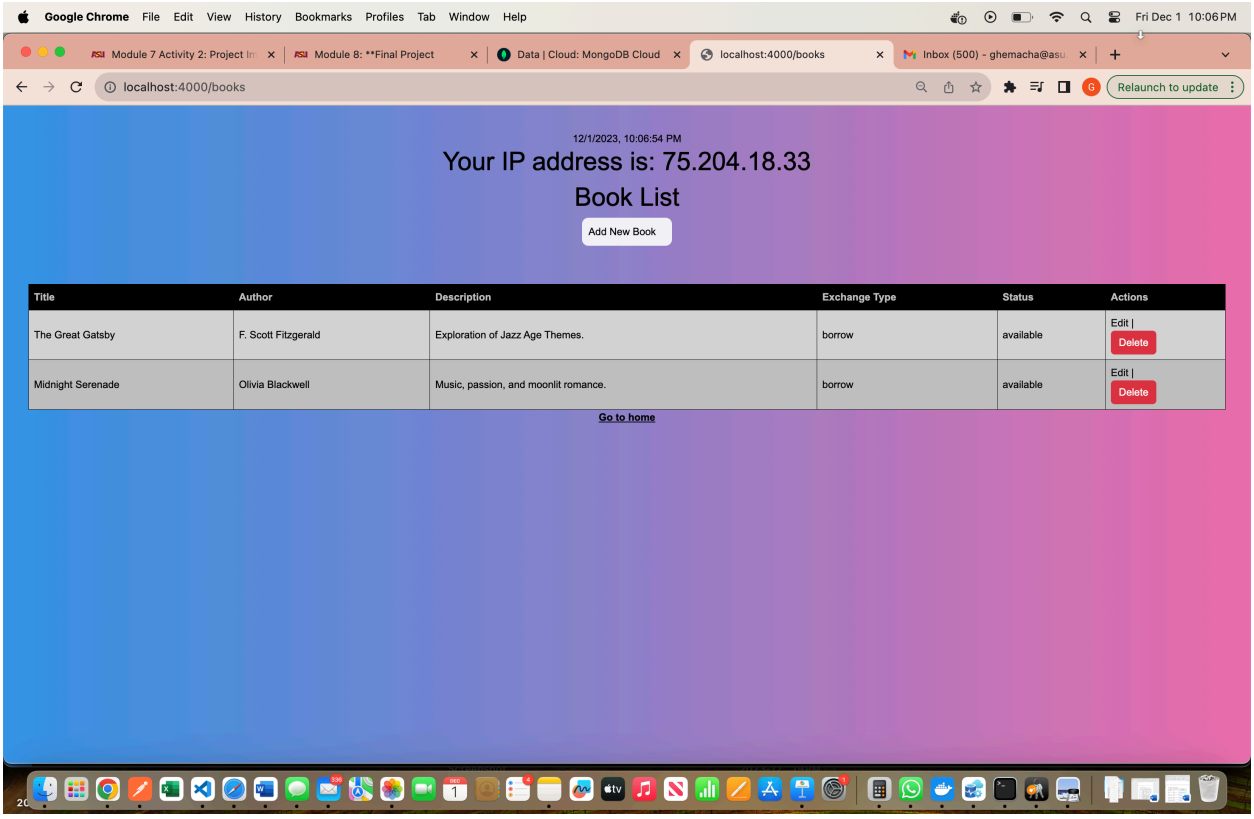
DELETE book 2 by ID:



There are 2 books as the 2nd book is deleted which was the 3rd book in database:



After all the CRUD operations the 2 books in the UI are:



Implementation

Discussion of challenges faced during the implementation phase, workarounds and solutions applied.

- Code Functionality: I faced issues with the code not working as expected, requiring thorough debugging and analysis of error messages to identify and resolve the root causes.

- Hard Refresh Necessity: The requirement for a hard refresh to see UI changes highlighted potential problems with cache headers and versioning. Adjustments were made to improve the development experience and streamline the refresh process.

- Unit Testing: Implementing comprehensive unit tests became crucial to catch unforeseen issues early in the development process and ensure the correctness of individual code components.

Key functionalities and features

Authentication Flow:

i. Sign-Up:

- Users initiate a sign-up process by requesting the creation of a new account.
- The server sends a sign-up form to the user.
- Users provide their sign-up data, which typically includes information like email and password.
- The server creates user data by storing the provided information in a database.

ii. Login:

- Users can choose to log in after successfully signing up.
- They request a login action.
- The server sends a login prompt to the user.
- Users submit their login credentials, which usually consist of an email and password.
- The server queries the database to validate the provided credentials.

- If the credentials are valid, the server generates an authentication token and sends it to the user, granting access.
- If the credentials are invalid, the server returns an authentication failure response.

Authorization Flow:

i. Access Request

- Users request access to certain functionalities or resources within the system.
- The server receives the access request.

ii. Authentication Check: Before proceeding, the server checks the authentication status of the user to ensure they are logged in.

iii. Authorization Check:

- If the user is authenticated, the server proceeds to check the user's authorization level.
- It queries the database to fetch the user's permissions and roles.

iv. Authorization Decision:

- Based on the user's permissions and roles, the server decides whether to grant or deny access to the requested resource or functionality.
- If the user has the necessary permissions, access is granted.
- If not, access is denied.

Security Measures:

i. Encryption:

- The system likely employs encryption techniques, especially during the login process. This ensures that user credentials (such as passwords) are securely transmitted and stored.
- Encryption can protect data in transit and at rest, making it difficult for unauthorized parties to access sensitive information.

ii. Tokenization:

- Tokenization is utilized during the authentication process. After successful login, the server generates an authentication token.
- This token serves as proof of the user's authentication and is typically included in subsequent requests to access protected resources.
- Tokens are an added layer of security, reducing the need to transmit sensitive credentials with every request.

iii. Database Security: The security of user data in the database is crucial. The system should implement measures such as secure password storage (hashing), access control, and proper database configurations to prevent data breaches.

iv. Role-Based Authorization:

- Authorization decisions are made based on the user's role and permissions.
- Role-based authorization restricts users from accessing resources or performing actions they are not authorized for.

v. Session Management: The system likely manages user sessions securely, ensuring that user sessions expire after a certain period of inactivity to prevent unauthorized access.

Learning Outcomes

Learning Outcomes (Modules 5 to 8):

1. Comprehensive Proficiency in Full Stack Development: Attaining mastery in full-stack development involved the integration of front-end (JavaScript), back-end (Node.js), and database (MongoDB) components.

2. Advanced Mastery of JavaScript: A focus on strengthening foundational JavaScript skills and exploring advanced concepts empowered the development of more efficient and robust code.

3. Advanced Proficiency in Node.js: Achieving advanced proficiency in Node.js required a comprehensive understanding of its event-driven, non-blocking architecture, utilized for creating scalable and asynchronous applications.

4. Expertise in MongoDB Management: Enhancing skills in MongoDB involved gaining proficiency in NoSQL database management, mastering data modeling using Mongoose, and executing efficient CRUD operations.

5. Implementation of JWT Security: An expansion of knowledge on JSON Web Tokens (JWT) emphasized their significance in secure authentication and authorization. The successful implementation of JWT in real-world scenarios contributed to enhanced security practices.

Non-Technical Achievements:

1. Consistent Engagement and Motivation: Throughout Modules 5 to 8, there was a demonstrated commitment to learning and a passion for staying engaged with new technologies.

2. Problem-Solving Excellence: The development of effective problem-solving skills, particularly in addressing challenges related to authentication, authorization, and asynchronous operations, was a notable achievement.

3. Continuous Learning Mindset: Cultivating a mindset of continuous learning involved recognizing the dynamic nature of full-stack development and embracing the constant opportunity for improvement and exploration.

4. Valuable Insights through Project-Based Learning: Embracing project-based learning as a valuable method to apply theoretical knowledge practically, the Book Exchange Project provided a hands-on experience and contributed to a deeper understanding of real-world application.

Constructive Feedbacks

1. **User Communication:**

- Feedback*:* It would be beneficial to mention how users are informed about successful logout actions or the process for initiating a password reset. Clear communication with users adds to the transparency and usability of the system.
- Suggestion*:* Provide information on how users are notified of successful logout and the steps involved in initiating and completing the password reset process.

2. **Error Handling:**

- Feedback*:* Consider addressing how the system handles potential errors or unexpected situations during the logout and password reset processes. Robust error handling contributes to a more user-friendly application.
- Suggestion*:* Briefly describe the error-handling mechanisms in place, such as informative error messages and steps taken to prevent misuse or exploitation.

References

JSON Web Tokens: https://jwt.io/

Node.js: https://nodejs.org/en

MongoDB: https://www.mongodb.com

Module Slides