Book Exchange Website

Sushmita Prafull Halasawade

IFT 544: Middleware Prog & Database Sec

Dinesh Sthapit

1 Dec 2023

Table of Contents

Abstract: **3**

Introduction: **3**

Overview of the project: 3
Problem Statement: 4
Scope and Limitations: 4
Technology Stack: 4

Project Design: **5**

Architecture and Design: 5
Security Flow: 6
Entity Relationship Diagram (ERD): 8
MVC Architecture: 10
API specification: 14

Screenshots: **29**

Implementation: **44**

Challenges Faced During Implementation: 44
Workarounds and Solutions Applied: 45
Key Functionalities and Features: 45

Learning Outcomes: **46**

Constructive Feedback: **47**

References: **49**

Abstract:

The "Book Exchange" website is a user-friendly online platform designed for book enthusiasts to exchange books with each other. Developed using Node.js, Express, and MongoDB, this website offers a seamless user experience with a clean and responsive interface. Its primary functionalities include user authentication, book management, and handling exchange requests. Users can easily sign up, log in, list books for exchange, browse books offered by others, and initiate exchanges. The project architecture is robust, featuring well-structured modules for user and book management. Each user's profile contains essential information such as name, email, and encrypted passwords, while the book profiles include details like title, author, and exchange status. Security is a key aspect, with JWTs implemented for secure authentication and bcrypt.js for password hashing. Middleware functions further enhance security by protecting routes that require authentication. The Book Exchange website is a secure, easy-to-use platform that helps people share books and enjoy reading together online.

Introduction:

Overview of the project:

The "Book Exchange" website is a new online platform that helps readers exchange books. It's like a digital marketplace where people can list, browse, and exchange books. The goal of this project is to encourage reading and sharing of books by creating a network where book lovers can find and share books without having to buy them.

Problem Statement:

The main problem that the "Book Exchange" website solves is that there is no good, easy-to-use platform for book lovers to exchange books. Many readers have books they don't need any more and want to find new books to read. But the options for doing this are often limited, inconvenient, or expensive. This project aims to fix these problems by providing a central, easy-to-use platform specifically for exchanging books.

Scope and Limitations:

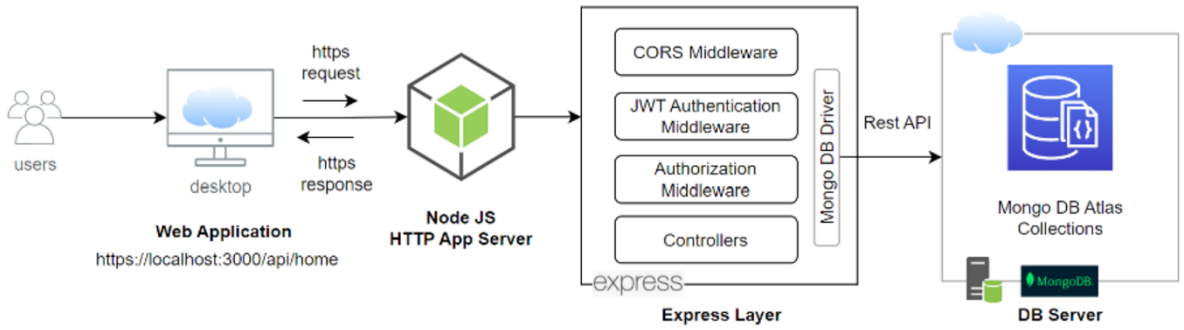
The scope of the project is to develop a full-fledged web application with features for user registration, book listing, and managing exchange requests. However, there are some limitations. The platform currently only works online, so you need internet access to use it. Additionally, the project does not handle the logistics or physical exchange of books, which is left to the users to coordinate. It relies on user honesty and accountability for the condition and delivery of books

Technology Stack:

This app uses Node.js, Express, and MongoDB technologies. Node.js provides a scalable environment for handling many connections at the same time, which is important for a community-driven platform. Express, a flexible Node.js web application framework, makes it easy to develop robust web features. MongoDB, a NoSQL database, is flexible and good at handling large amounts of unstructured data, making it suitable for the dynamic and growing collection of user and book information. This technology stack ensures a reliable, scalable, and user-friendly application, which aligns perfectly with the project's needs.

Project Design:

Architecture and Design:



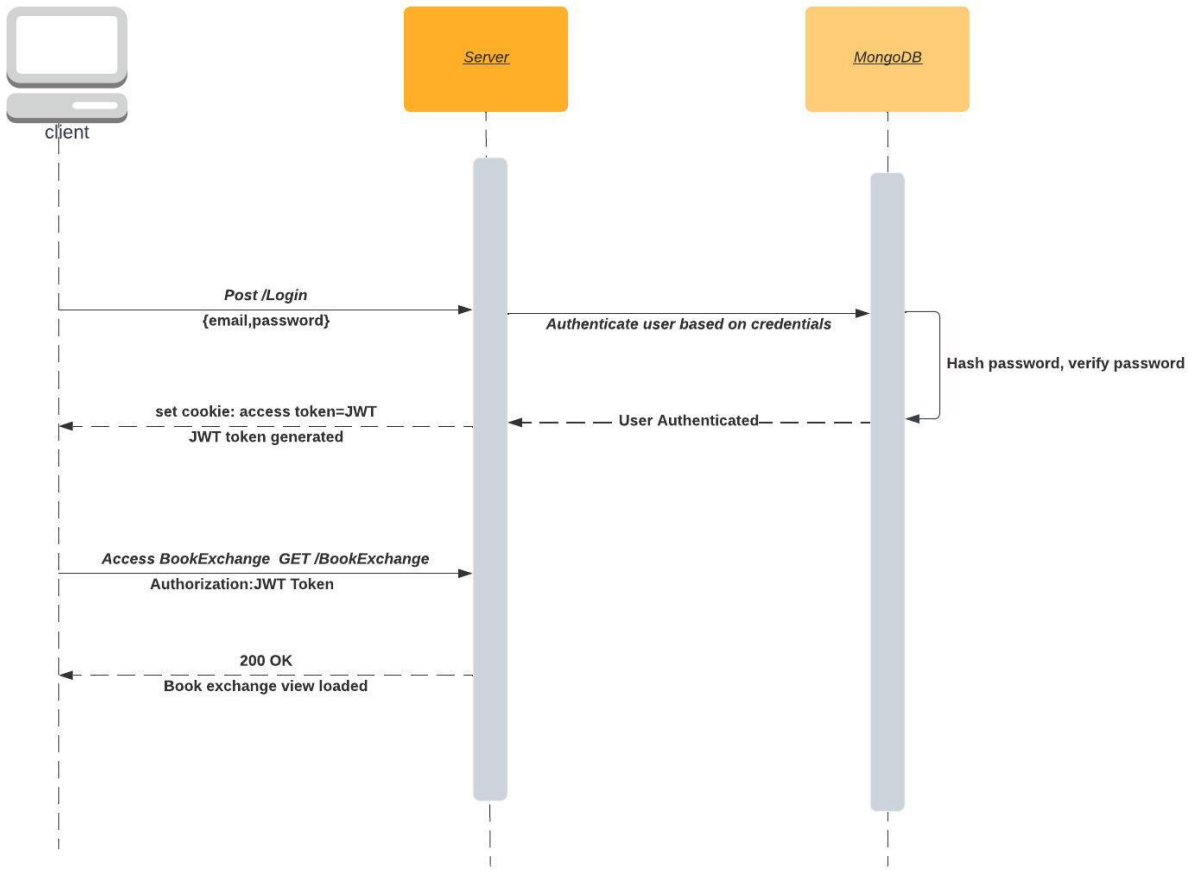
- **User Interaction:** User interacts with the web application through a browser or client-side application.
- **HTTP Request Processing:** The user's HTTP request is received by the Node.js HTTP application server.
- **Express Layer:** The request is then routed to the Express layer, which acts as a middleware framework.
- **Within Express:** CORS middleware handles Cross-Origin Resource Sharing to manage access control JWT authentication verifies and authenticates user identity based on JSON Web Tokens, Authorization logic determines access rights before proceeding, Controller functions handle the business logic and orchestrate data processing.
- **Database Interaction:** RESTful API requests, formed by the Express layer, are directed to the database server, MongoDB driver facilitates communication between the Node.js

application and the MongoDB database. Database operations are performed based on the received RESTful API requests.

- **Data Processing:** The database server processes the received requests, executing operations such as CRUD (Create, Read, Update, Delete) in the MongoDB database.
- **HTTP Response Handling:** After processing, the response is sent back through the same flow. The response travels from the database server to the Node.js HTTP application server.

Security Flow:

Authentication and authorization with JWT (JSON Web Tokens Introduction, n.d.):

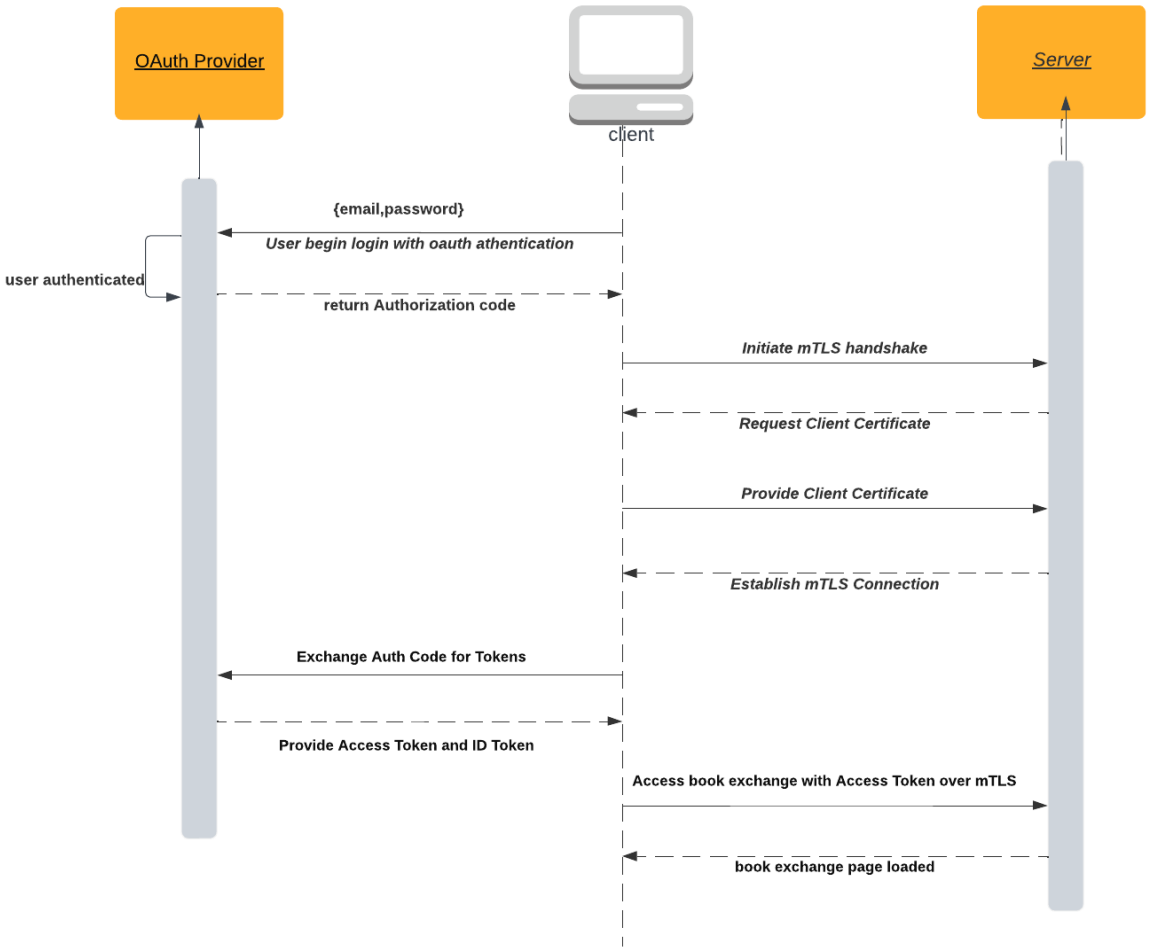


•User logs in with email and password; JWT token is generated and set as a cookie.

•Authenticated user accesses the book exchange page displaying available book details.

•JWT token is sent with each request, allowing the server to authenticate and authorize the user.

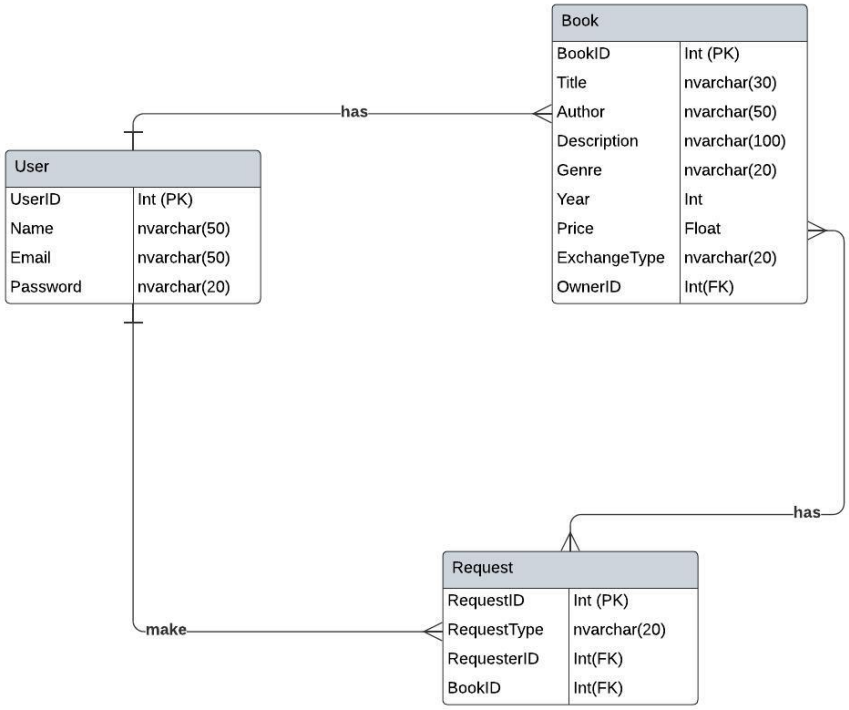
Authentication and Authorization process after oauth (oAuth, 2020)and mtls(Porter, 2021):



- User selects "Login via OAuth Provider" on the client application.
- OAuth server presents authentication interface to the user.

- User enters credentials and authenticates with the OAuth provider.
- OAuth server redirects back to the client with an authorization code.
- Client exchanges authorization code for an access token and an ID token.
- OAuth server sends the tokens back to the client.
- Client initiates a TLS handshake with the server.
- Server requests the client's certificate.
- Client presents its certificate.
- Server authenticates the client's certificate and establishes an mTLS connection.
- Client sends a request to the resource server with the access token over the mTLS connection.
- Resource server validates the token and serves the request.

Entity Relationship Diagram (ERD):



User Entity: Represents the registered users of the application. Each user can own multiple books.

Attributes: UserID (Primary Key), Name, Email, Password

Relationships: One-to-Many with Book Entity (One user can have multiple books)

Book Entity: Represents the books available for exchange. Each book is associated with a user (owner). It also has a Many-to-Many relationship with the Request Entity, indicating that a book can have multiple requests.

Attributes: BookID (Primary Key), Title ,Author ,Description ,Genre ,Year ,Price, ExchangeType (Borrow/Trade) , OwnerID (Foreign Key referencing UserID from User Entity)

Relationships:Many-to-One with User Entity (Many books can belong to one user), Many-to-Many with Request Entity (Many books can have many requests)

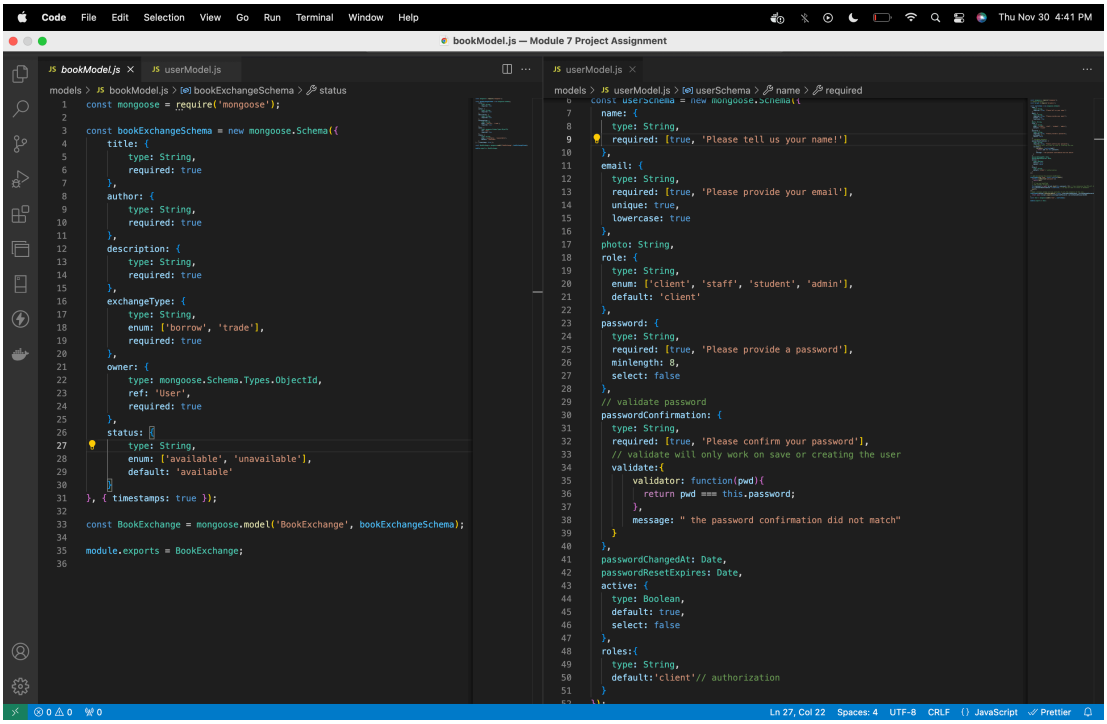
Request Entity: Represents the requests made by users for borrowing or trading books. It includes information about the type of request, the requester, and the book being requested.

Attributes: ,RequestID (Primary Key) ,RequestType (Borrow/Trade) ,RequesterID (Foreign Key referencing UserID from User Entity) ,BookID (Foreign Key referencing BookID from Book Entity)

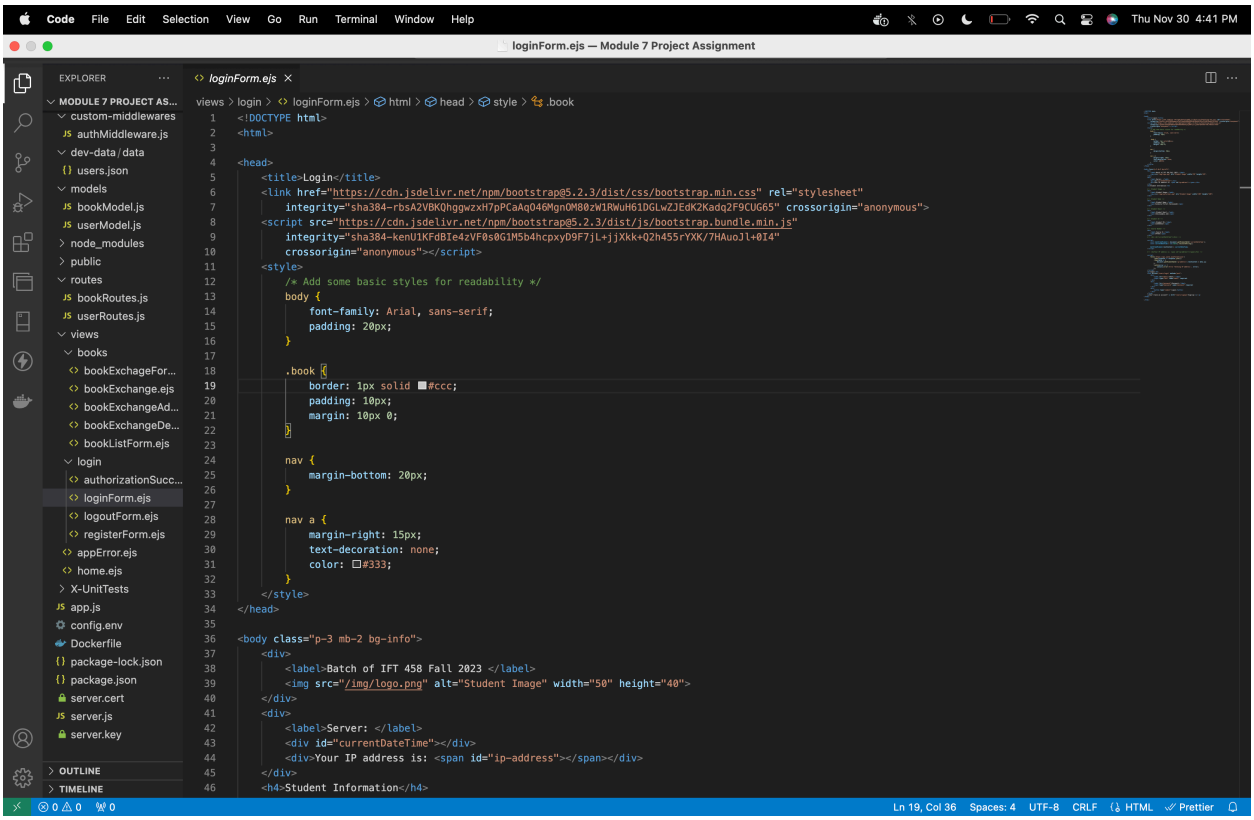
Relationships:Many-to-One with User Entity (Many requests can be made by one user), Many-to-Many with Book Entity (Many requests can be made for one book)

MVC Architecture:

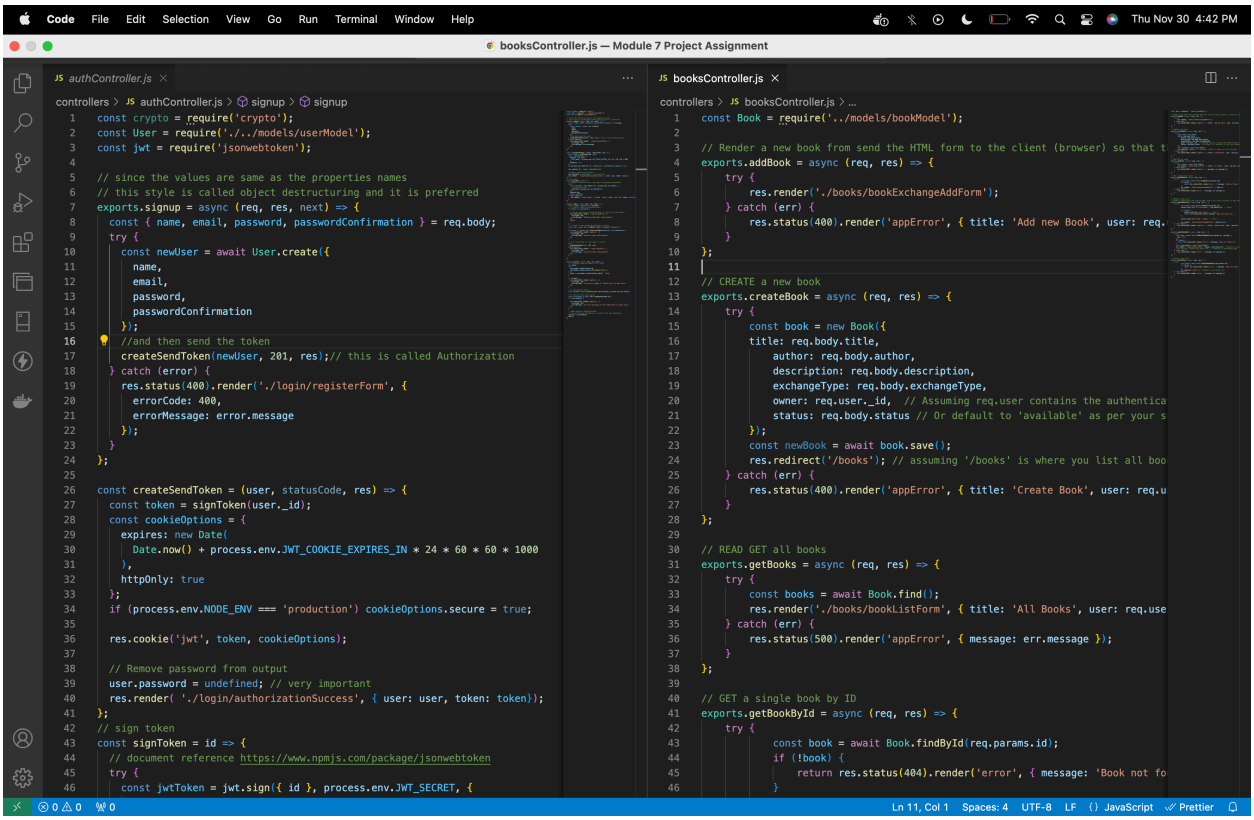
- **Model:** The models define the data structure. In this project, there are two primary models: User and BookExchange. The User model includes fields like name, email, password, while the BookExchange model has fields for title, author, description, exchange type, owner, and status. These models are defined using Mongoose (Mongoose, 2011), a MongoDB object modeling tool, allowing for easy interaction with the database.



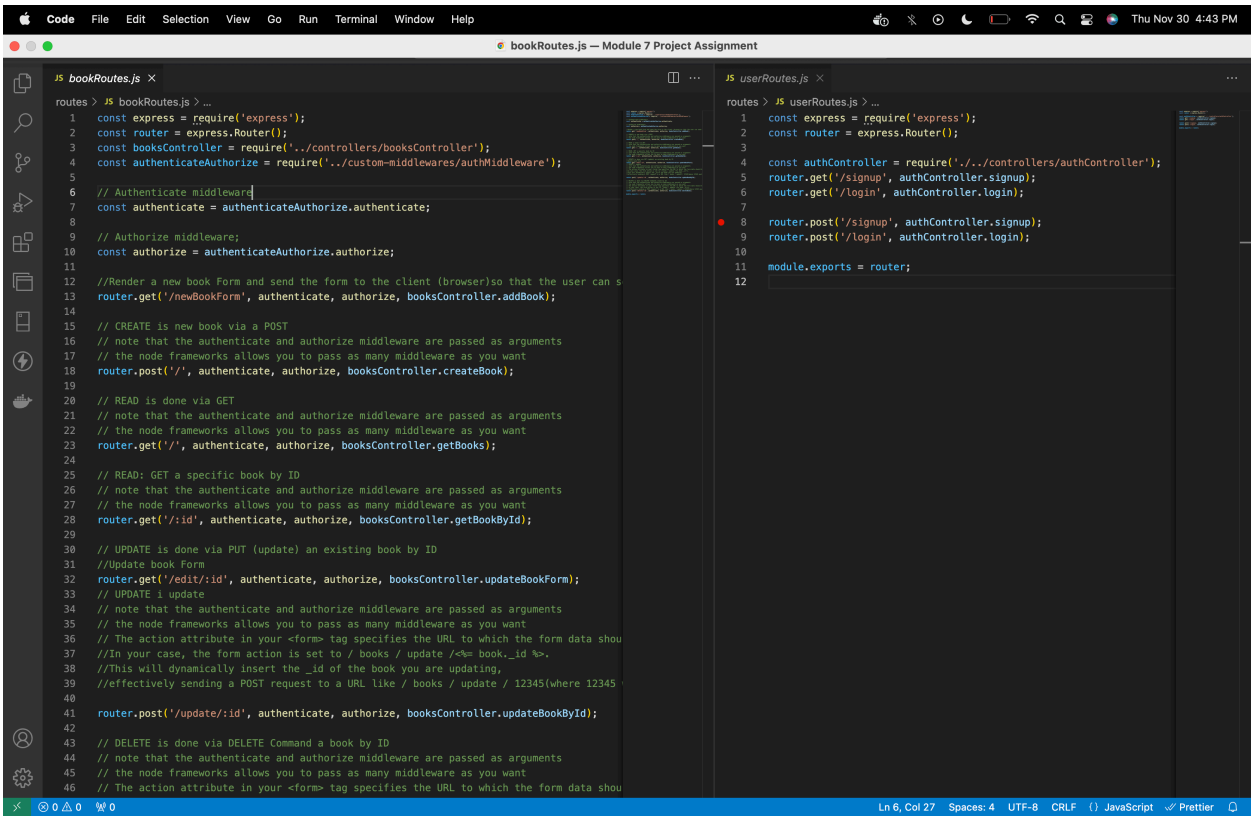
- **View:** Views are used to render the user interface. This project uses .ejs (*Ejs*, 2018) templates for views, which allow embedding JavaScript into HTML templates. These views are rendered based on the data received from the controllers and are displayed to the user. For instance, there are views for listing books, adding new books, and user authentication processes.



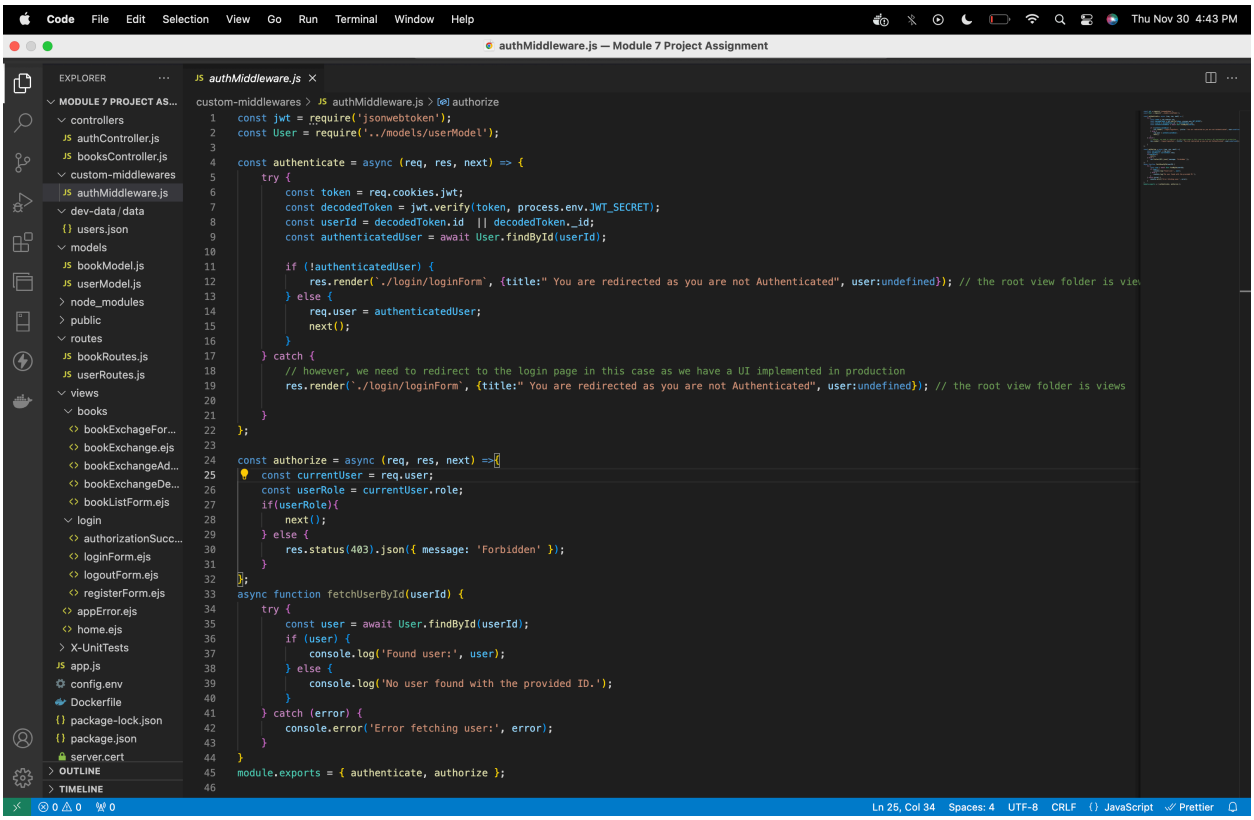
- **Controller:** Controllers handle the business logic. This codebase includes controllers like authController for handling user authentication and booksController for managing book-related operations. For example, booksController contains functions for adding, creating, updating, and deleting books, as well as fetching book details.



- **Routes:** The website's routing is handled using Express, a web application framework for Node.js. Routes are defined to respond to various HTTP requests (GET, POST, PUT, DELETE) corresponding to different functionalities. For instance:Routes related to books (bookRoute) include paths for creating a new book (/newBookForm), adding a book (/add), and getting book details (/:id).User-related routes (userRoute) handle user authentication, including signup (/signup) and login (/login).



- **Middleware:** Middleware functions are used to execute code, modify the request and response objects, and end the request-response cycle. This project uses middleware for various purposes:Authentication Middleware (authMiddleware) checks if the user is authenticated and authorized to access certain routes. It includes JWT verification to ensure that the user has a valid token. Middleware functions are used in routes to protect certain endpoints. For example, routes for adding or editing books include authentication and authorization middleware to ensure only logged-in users can access these functionalities.



API specification:

Feature	Method	URL End Point
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 Form	GET	http://localhost:4000/books/newBookForm

Add Book Exchange	POST	http://localhost:4000/books
Get book by ID	GET	http://localhost:4000/books/:id
Update Book Exchange Form	GET	http://localhost:4000/books/edit/:id
Update Book Exchange	POST	http://localhost:4000/books/update/:id
Delete Book Exchange	POST	http://localhost:4000/books/delete/:id

1. **Register / Sign up:**

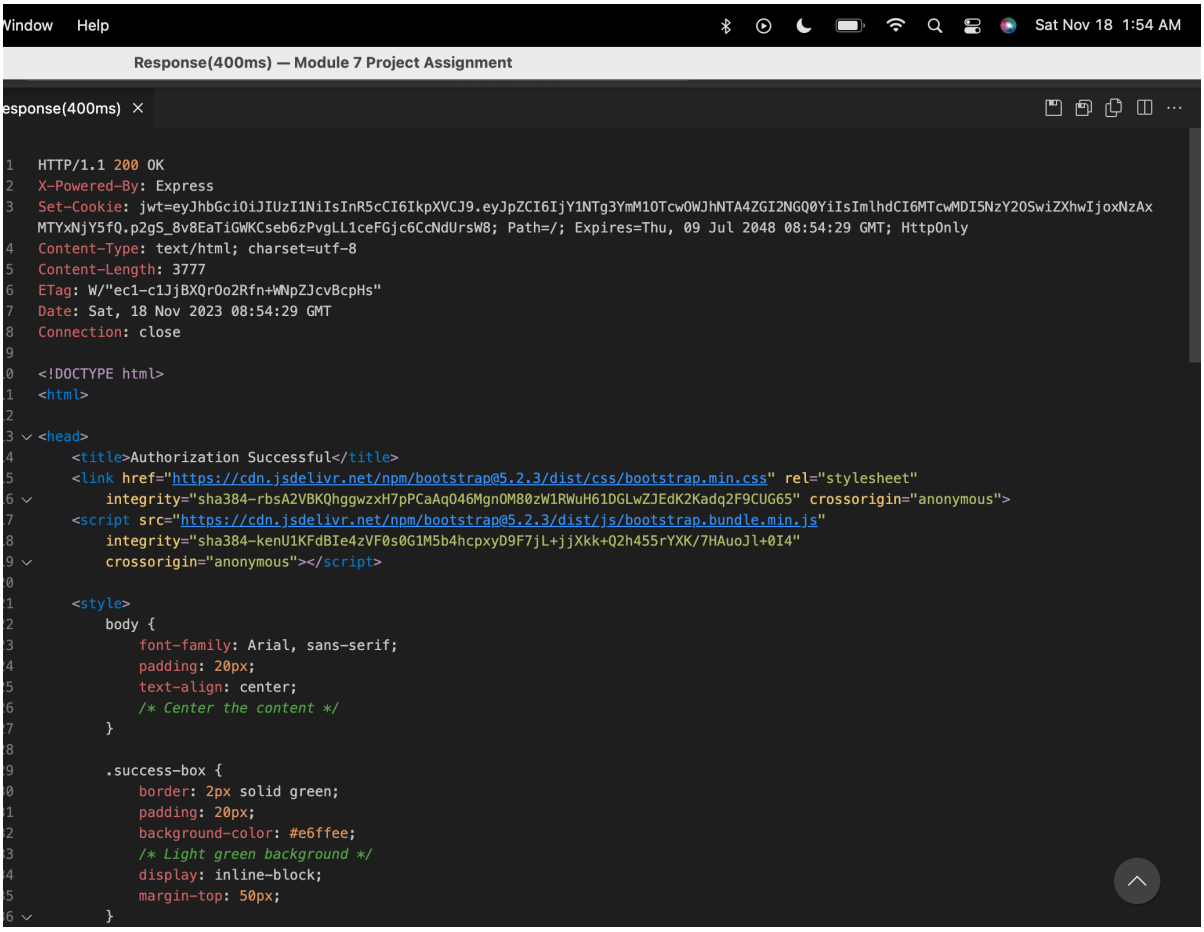
Request:

```
### Sample Test: register user

POST http://localhost:4000/users/signup

Content-Type: application/json



{

  "name": "Sushmita Prafull Halasawade",

  "email": "shalasaw@asu.edu",

  "password": "Password@1234",

  "passwordConfirmation": "Password@1234"
```

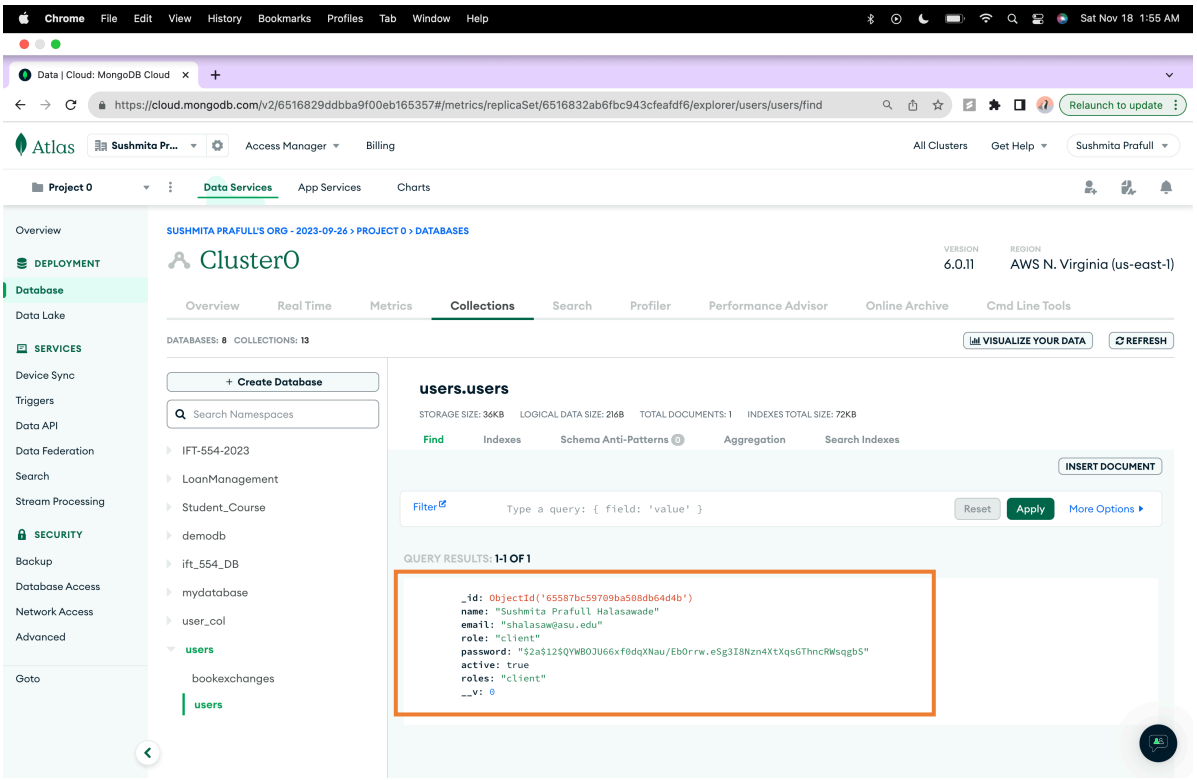
```
}
```

Upon a successful POST to '/signup', the server validates the data, creates a new user, issues a JWT as a secure cookie, and displays a success view with user details, sans the password. In case of errors, such as token issues or validation failures, it renders the login view with an appropriate error message.

Response:



Database:After user creation: user data is saved in users collection



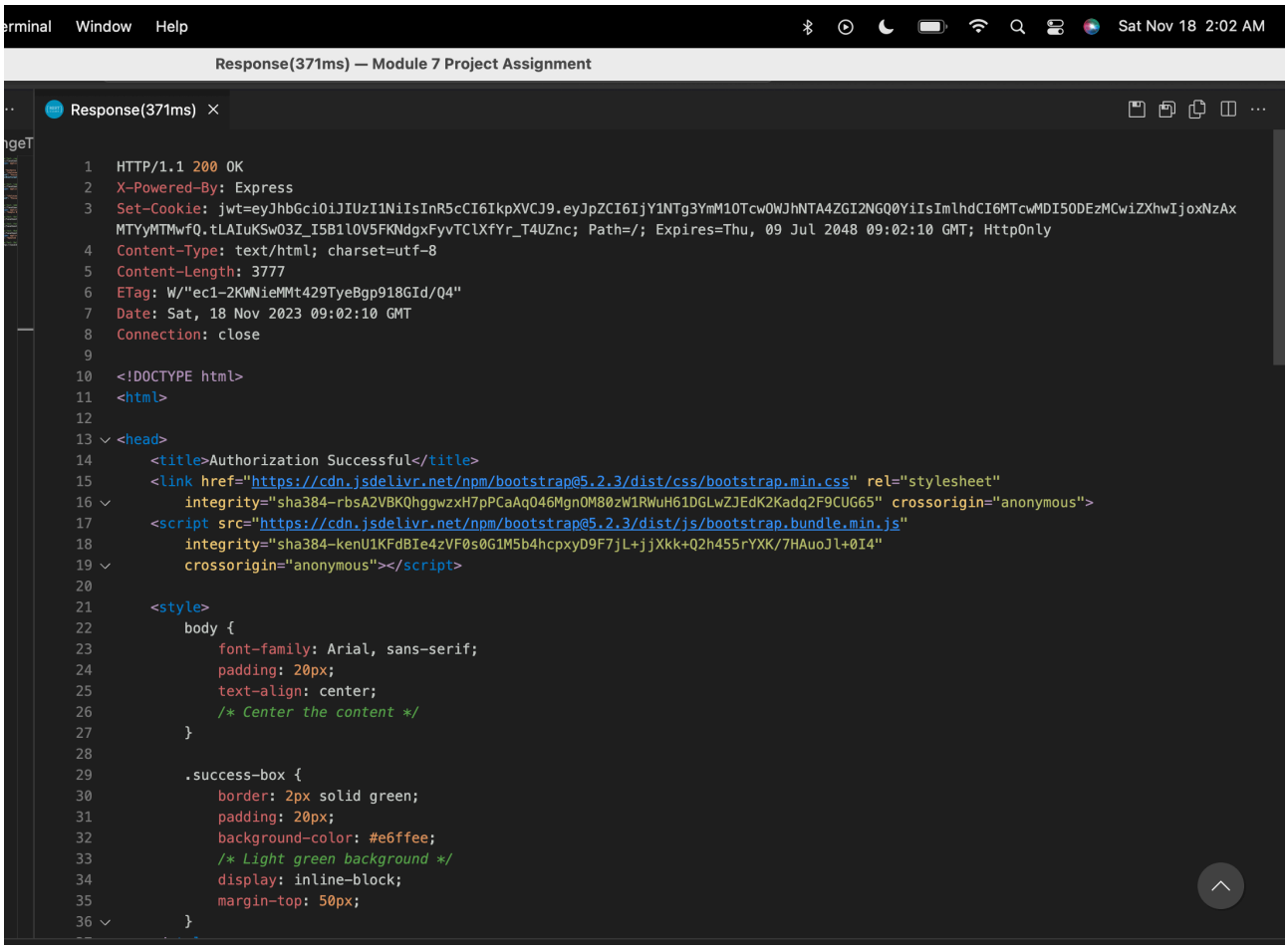
2. Login user:

Request:

```
### Sample Test: login user

POST http://localhost:4000/users/login

Content-Type: application/json

{

  "email": "shalasaw@asu.edu",

  "password": "Password@1234"

}
```

"/login" endpoint in the Book Exchange website performs user authentication by verifying the provided email and password against the database. If authentication fails due to an incorrect email, password, or an error, it displays an error message on the login form; upon success, it generates and sends a token for further authentication.

Response: jwt token



3. **Add book:**

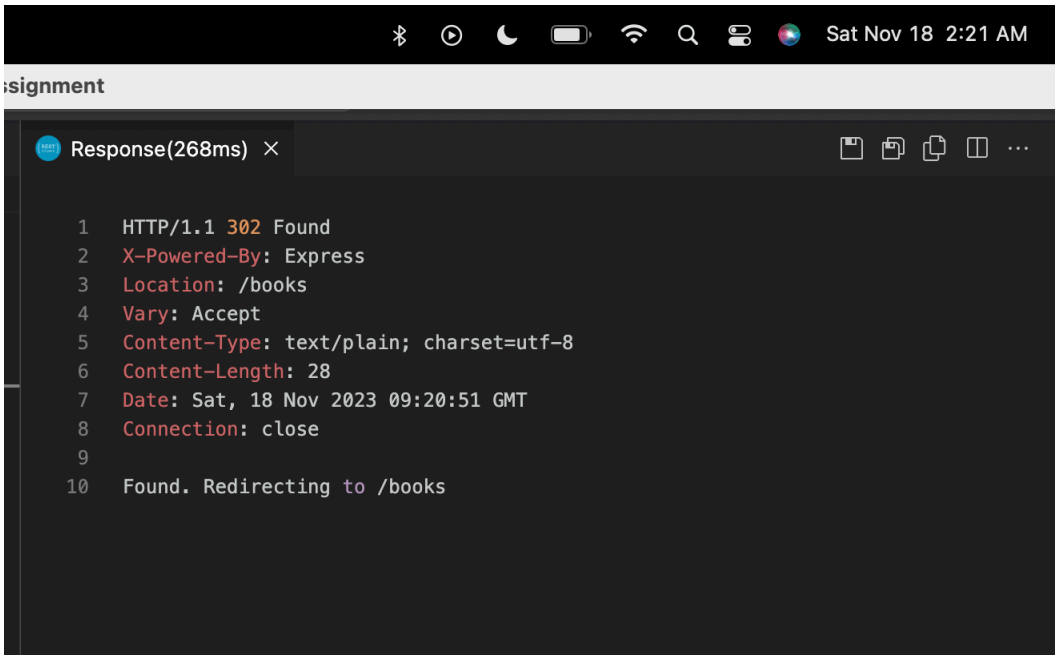
Request:

```
### Sample Test: Add a new book
```

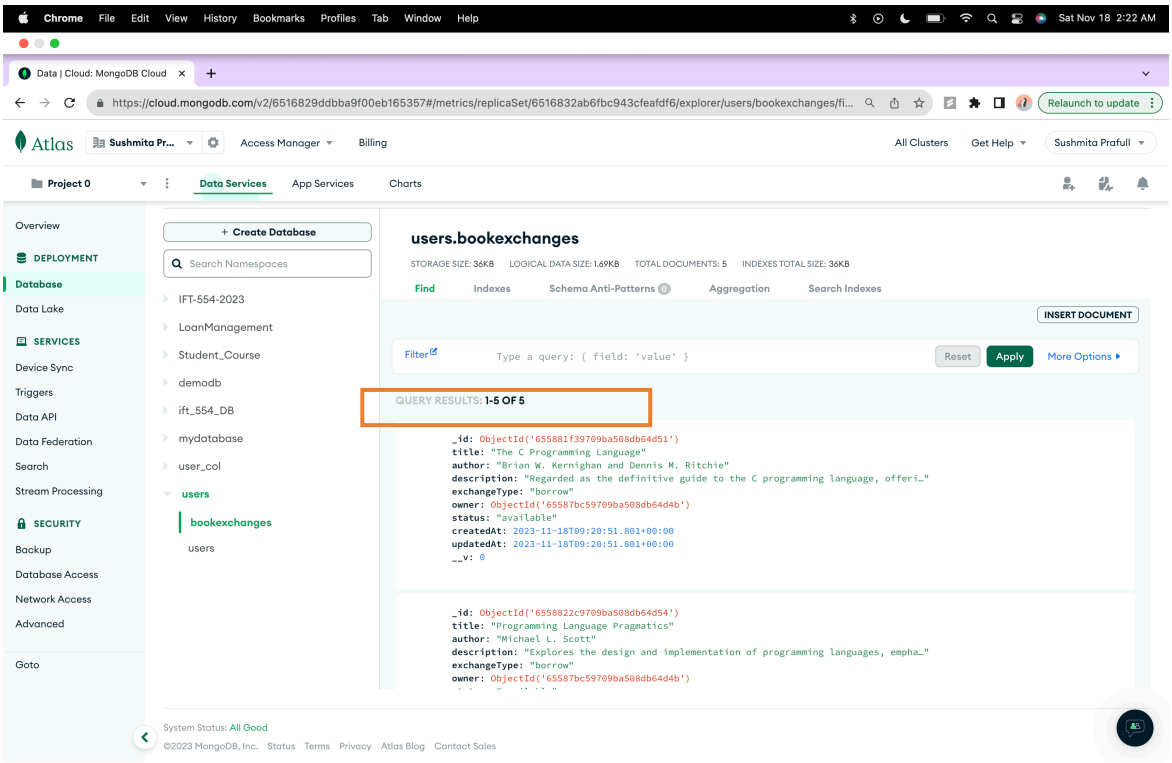
```
POST http://localhost:4000/books

Content-Type: application/json

Cookie:

jwt={eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTg3YmM1OTcwOWJhNTA4

ZGI2NGQ0YiIsImlhdCI6MTcwMDMzODY2MCwiZXhwIjoxNzAxMjAyNjYwfQ.WceXRkym

I058J-f1BwcsP8YQCd5wOxAoZlTLF5j4zGA}


{ "title": "The C Programming Language",

 "author": "Brian W. Kernighan and Dennis M. Ritchie",

 "description": "Regarded as the definitive guide to the C programming language, offering

concise and practical insights into programming with C",

"exchangeType": "borrow",

"owner": "65587bc59709ba508db64d4b",

"status": "available" }
```

the website creates a new Book object using request body data, including title, author, and owner, with access restricted by JWT-based authentication middleware. On successful book creation, it redirects to '/books', or displays an error page for any validation issues encountered.

Response: book created and redirected to /books



Database:After addition of books:



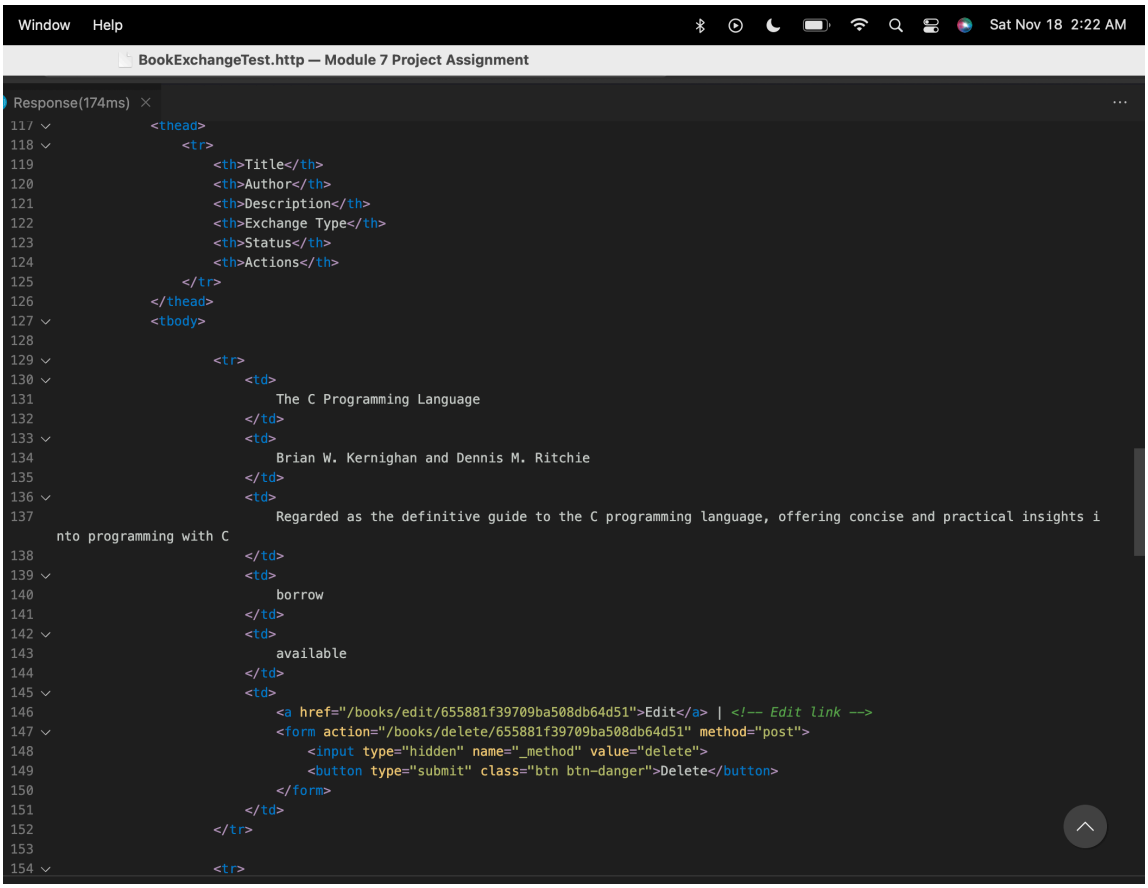
4. **Retrieve all books:**

Request:

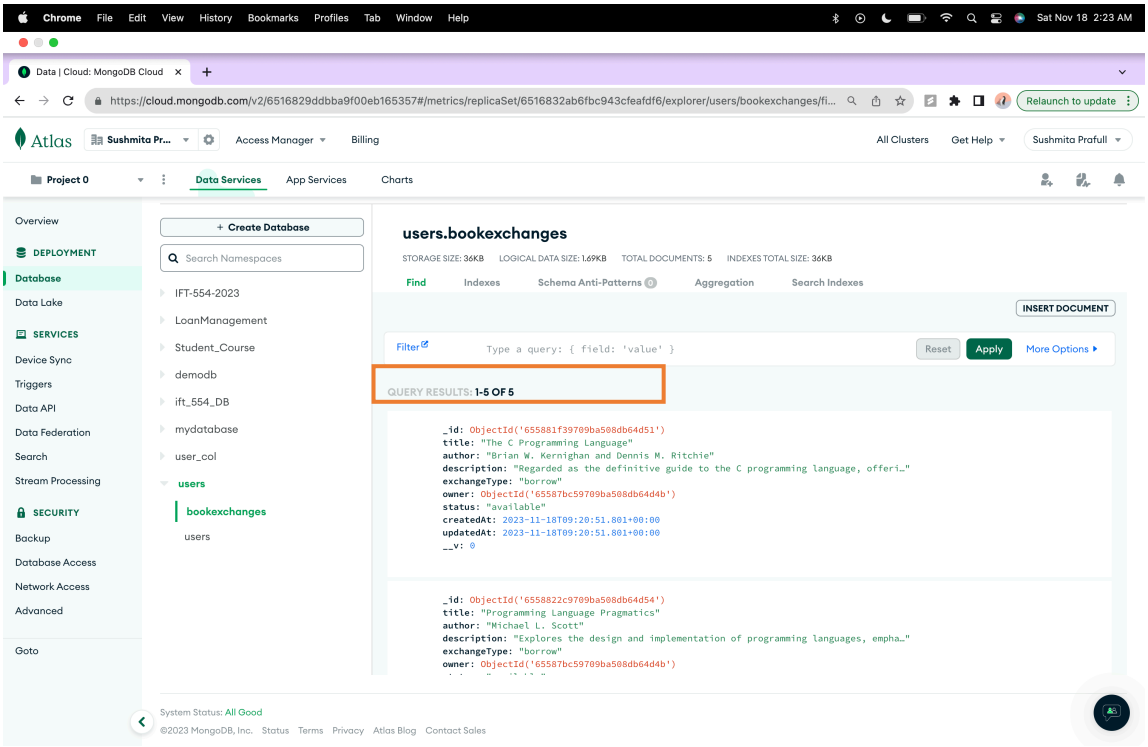
```
GET http://localhost:4000/books

Cookie:

jwt={eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTg3YmM1OTcwOWJhNTA4

ZGI2NGQ0YiIsImlhdCI6MTcwMDMzODY2MCwiZXhwIjoxNzAxMjAyNjYwfQ.WceXRkym

I058J-f1BwcsP8YQCd5wOxAoZlTLF5j4zGA}
```

The GET request route /books uses authentication and authorization middleware to verify JWT tokens in cookies, then calls getBooks, an asynchronous function that fetches all books from the database using Book.find(). If successful, it renders bookListForm with 'All Books', user info, and book data; if there's an error, it returns a 500 status and displays an appError view with the error message.

Response: 'bookListForm'



Database:



5. Get book by ID:

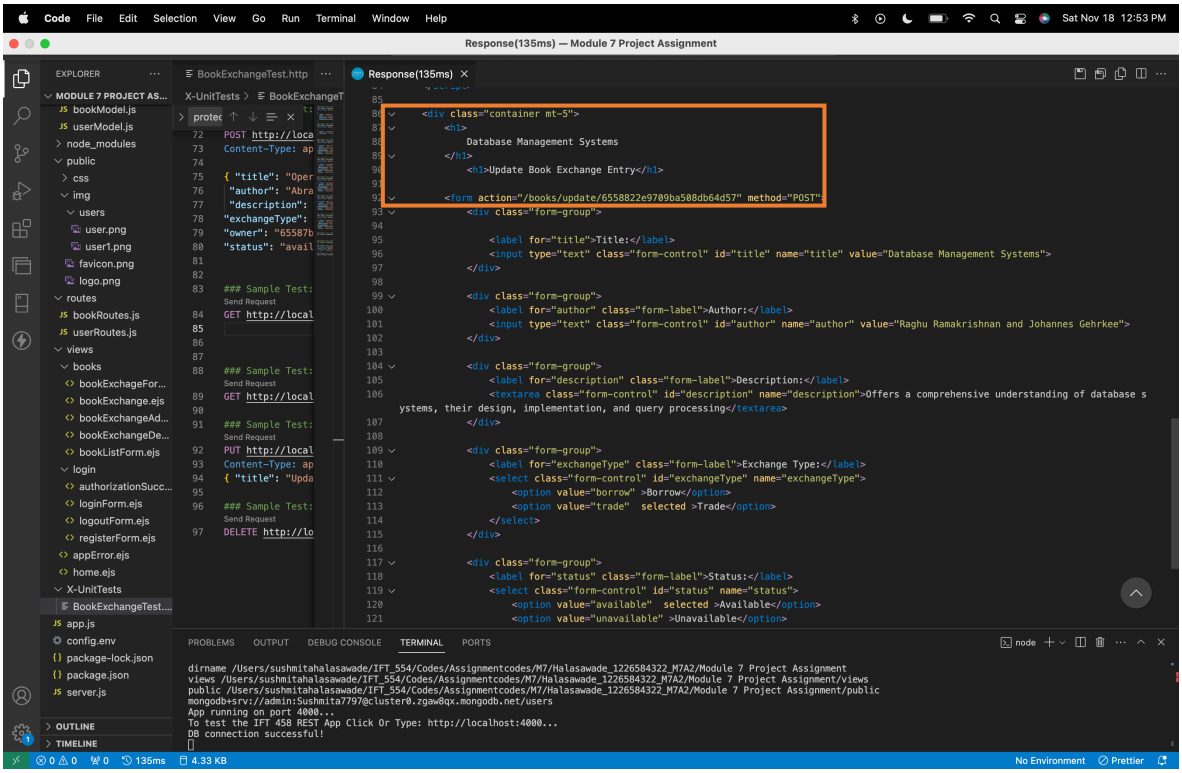
Request:

```
### Sample Test: Fetch book by ID

GET http://localhost:4000/books/6558822e9709ba508db64d57

Cookie:

jwt={eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTg3YmM1OTcwOWJhNTA4

ZGI2NGQ0YiIsImlhdCI6MTcwMDMzODY2MCwiZXhwIjoxNzAxMjAyNjYwfQ.WceXRkym

I058J-f1BwcsP8YQCd5wOxAoZlTLF5j4zGA}
```

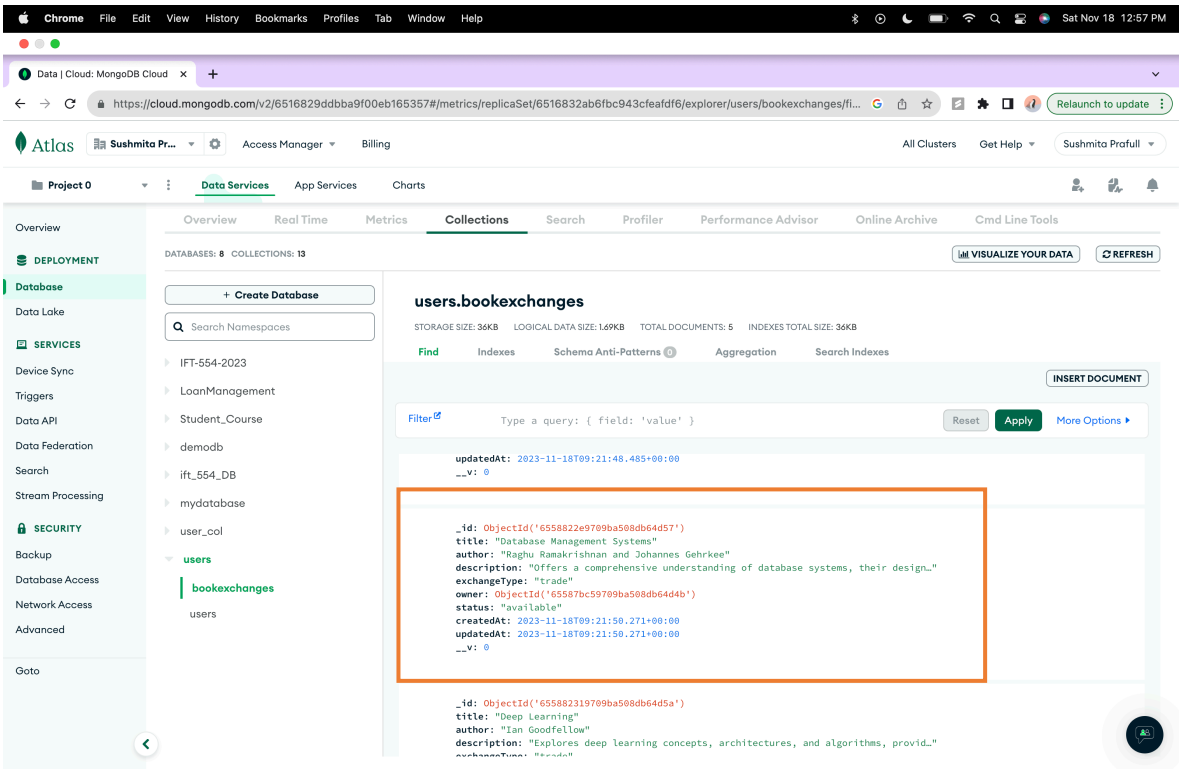
The getBookById function asynchronously retrieves a book from the database using its ID, secured by authentication and authorization middleware using JWT in cookies. It renders book

exchange details if found, or returns a 404 error for missing books, and a 500 status for other errors.

Response: book exchange details



Database:



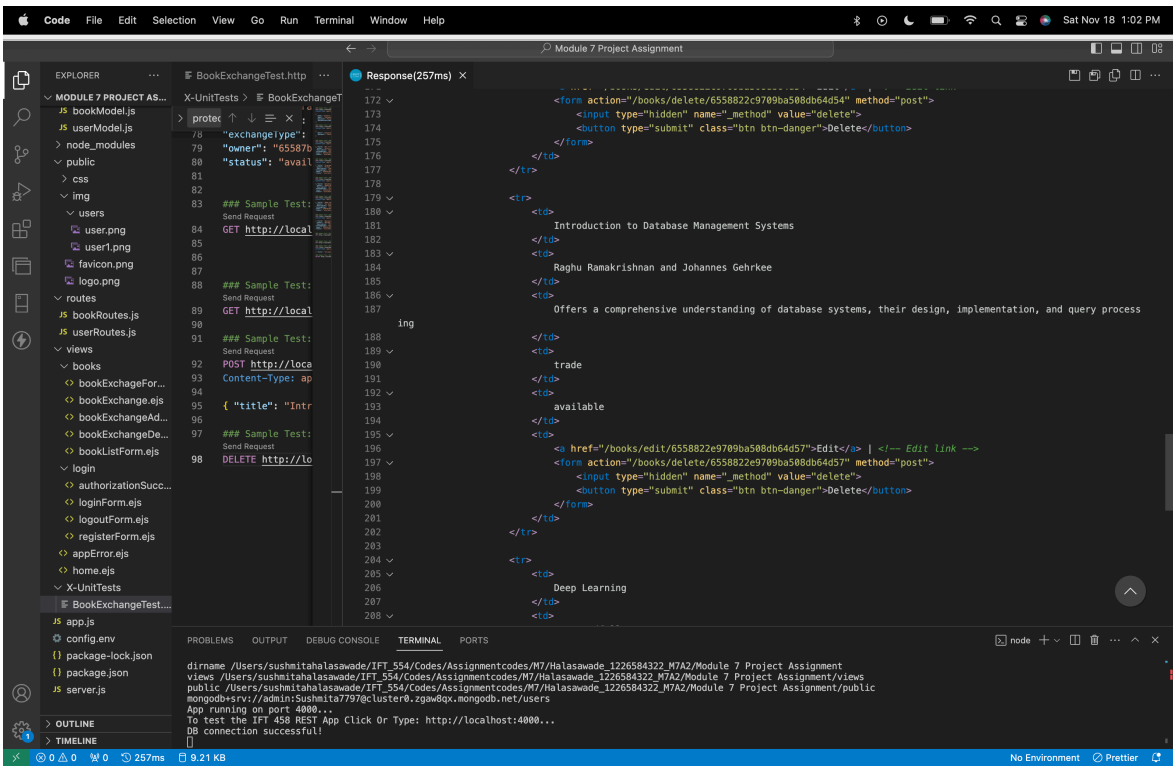
6. Update book:

Request:

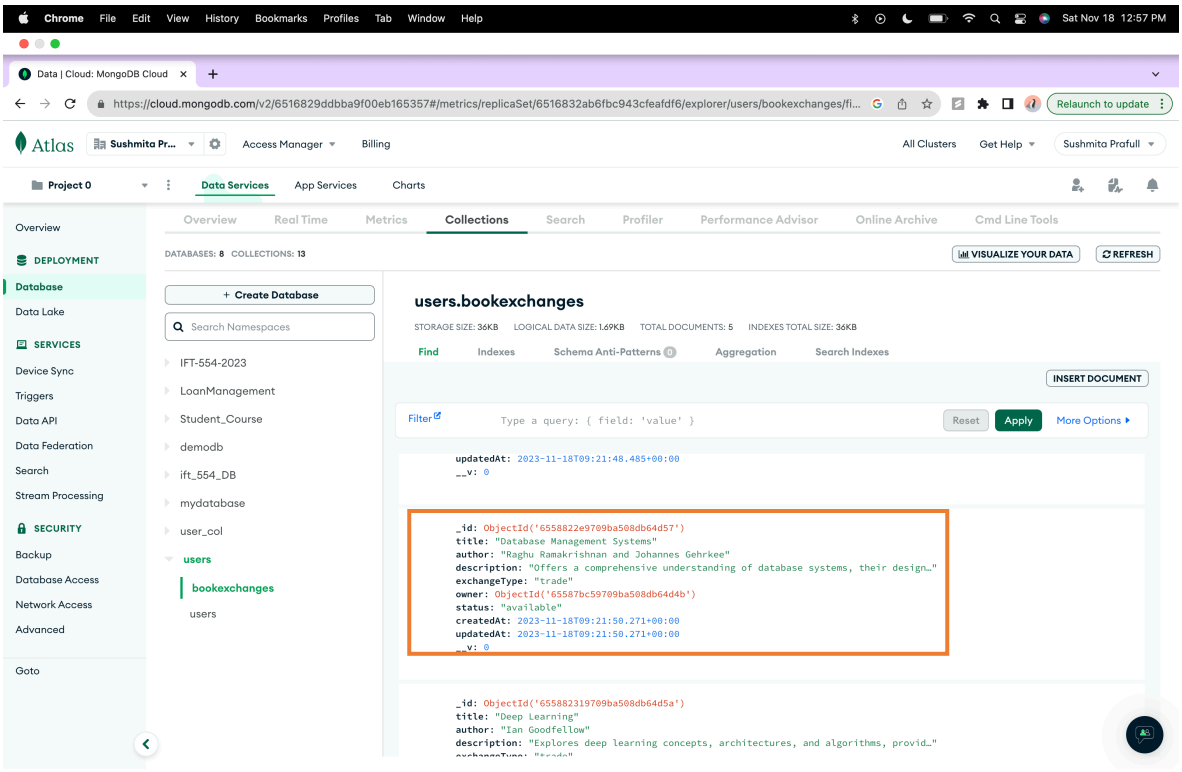
```
### Sample Test: Modify a book by ID

POST http://localhost:4000/books/update/6558822e9709ba508db64d57

Content-Type: application/json

Cookie:

jwt={eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTg3YmM1OTcwOWJhNTA4

ZGI2NGQ0YiIsImlhdCI6MTcwMDMzODY2MCwiZXhwIjoxNzAxMjAyNjYwfQ.WceXRkym

I058J-f1BwcsP8YQCd5wOxAoZlTLF5j4zGA}



{ "title": "Introduction to Database Management Systems" }
```

The updateBookById function in the Book Exchange website updates a book using its ID and req.body data with the findByIdAndUpdate method. On success, it shows the updated book list, while failures lead to a 404 error page or a 500 status with an error message in case of other errors.

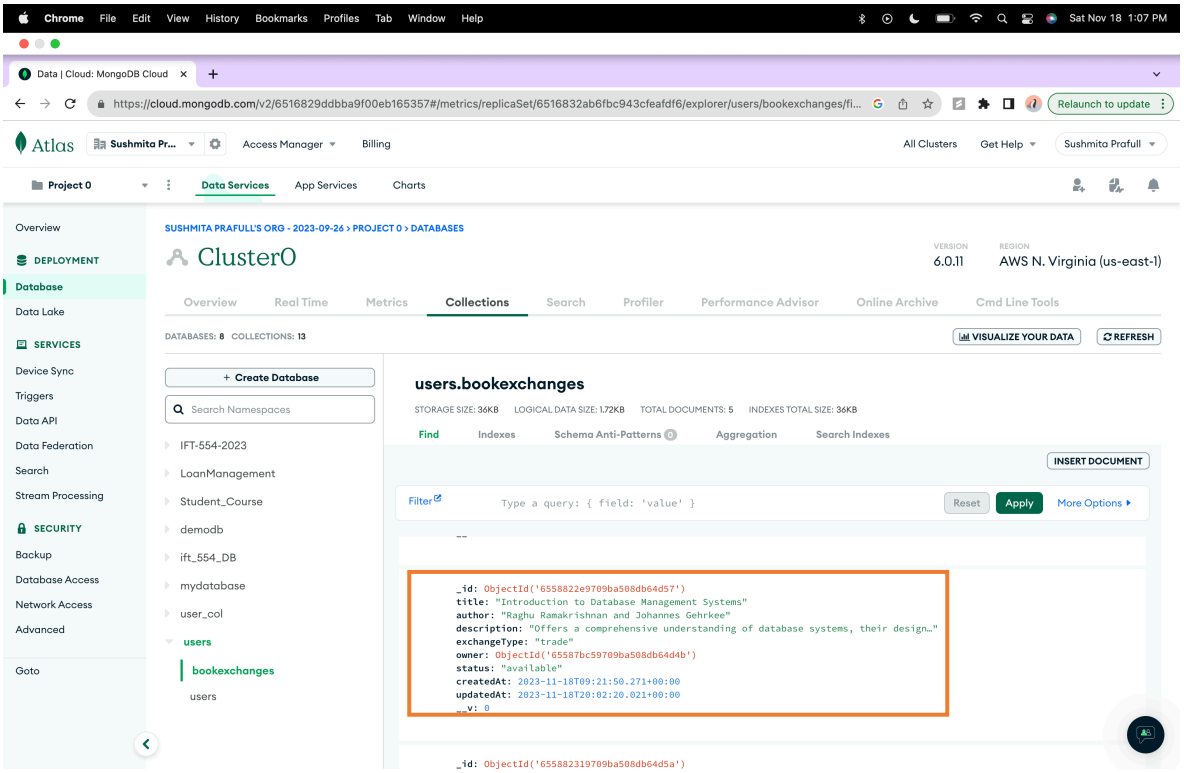
Response: book list



Database:Before updating database management book:



After updating the title intro to database management:



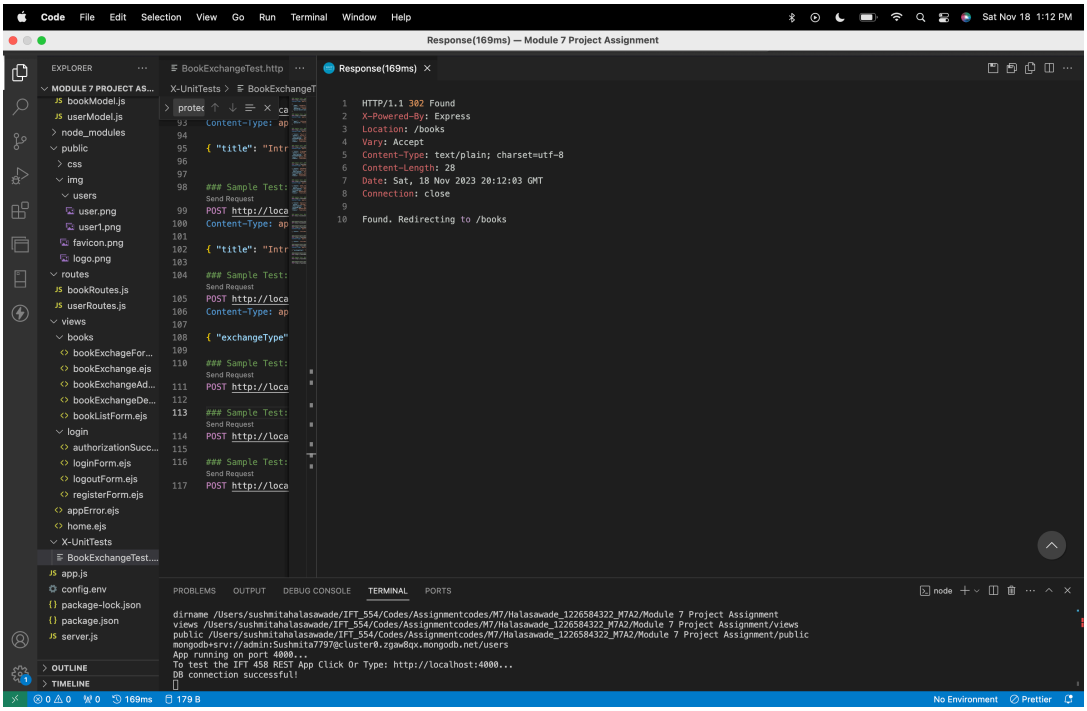
7. **Delete book:**

Request:

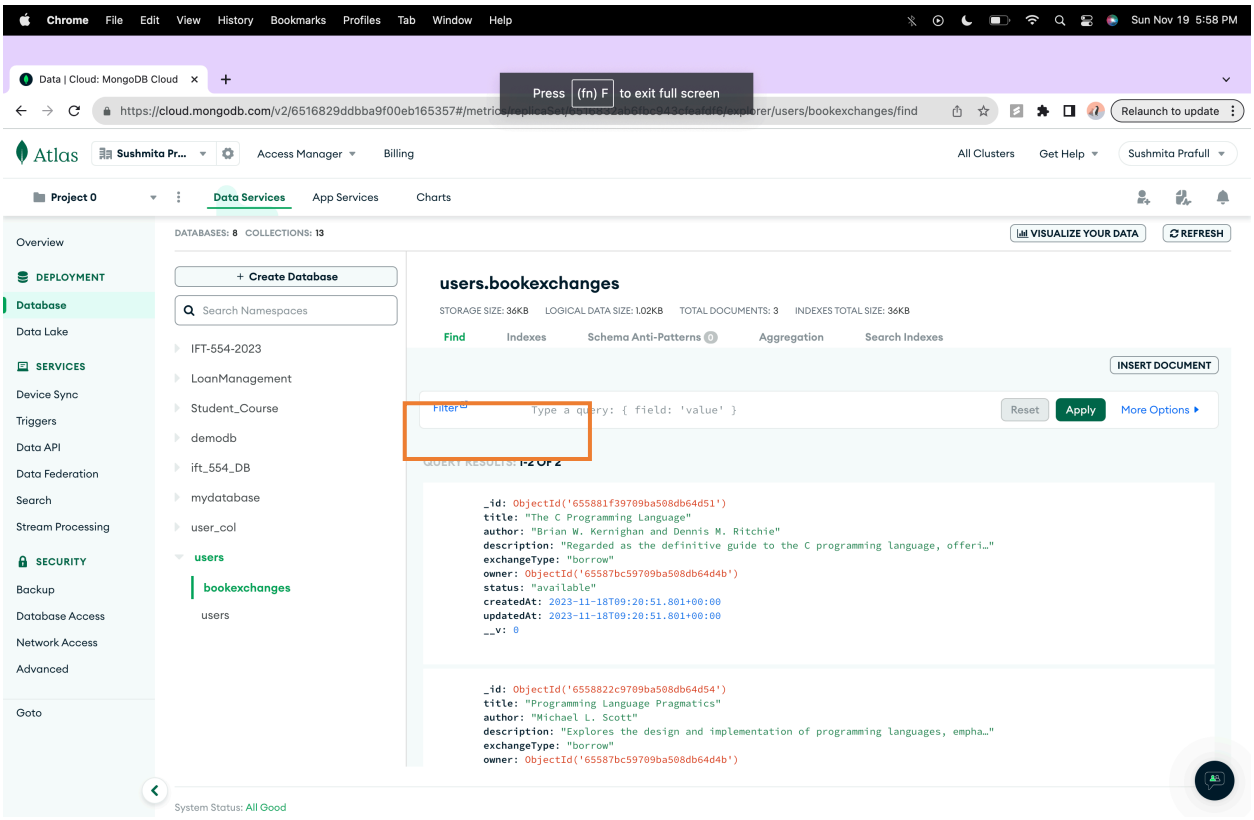
```
### Sample Test: Delete book by ID

POST http://localhost:4000/books/delete/6558822e9709ba508db64d57

Cookie:

jwt={eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTg3YmM1OTcwOWJhNTA4

ZGI2NGQ0YiIsImlhdCI6MTcwMDMzODY2MCwiZXhwIjoxNzAxMjAyNjYwfQ.WceXRkym

I058J-f1BwcsP8YQCd5wOxAoZlTLF5j4zGA}
```

The deleteBook function uses findByIdAndRemove to delete the book by its ID, redirecting to '/books' upon success, or rendering a 404 error if not found. Protected by authentication and authorization middleware, it validates user access via a JWT token stored in a cookie.

Response:

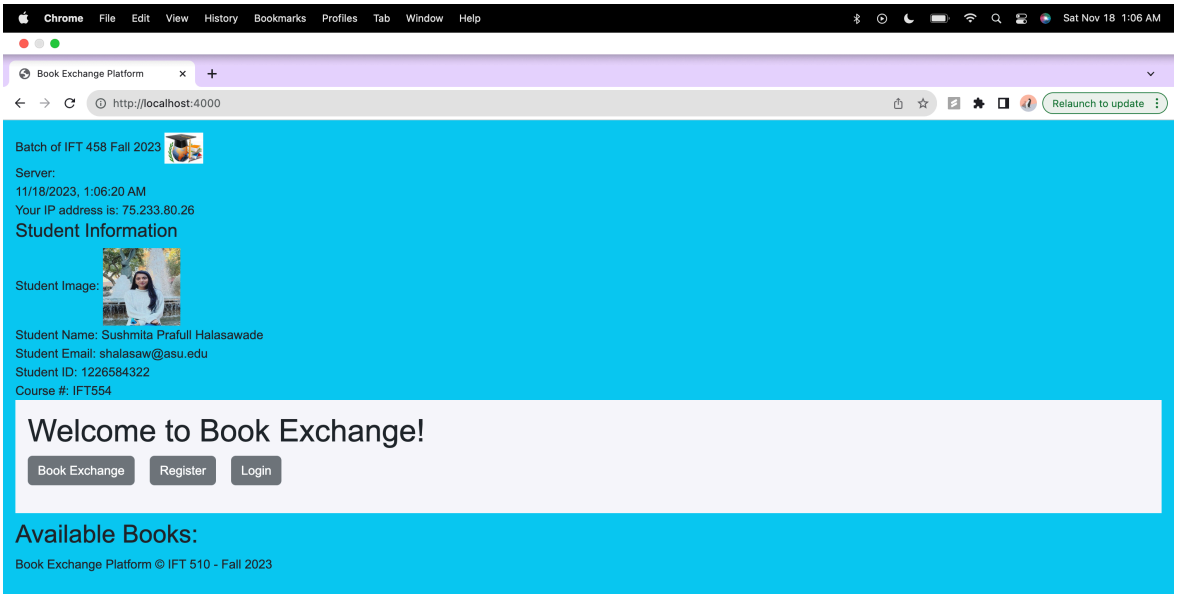


After deleting books:

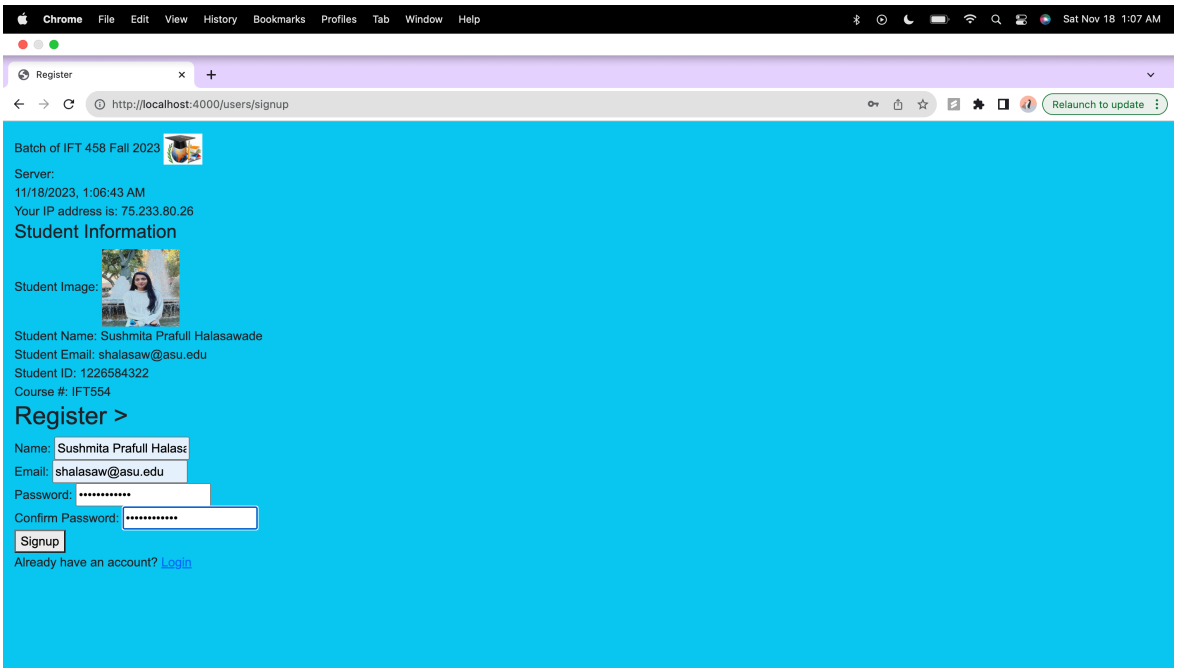


Screenshots:

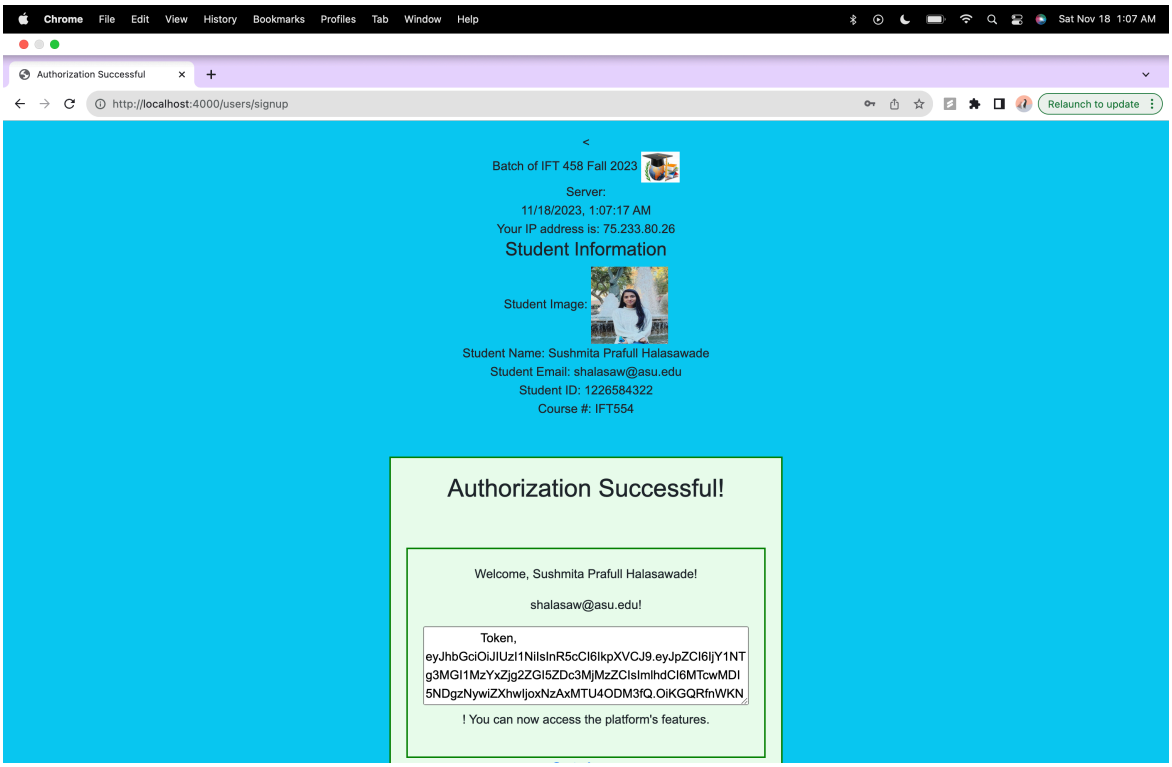
Home page:



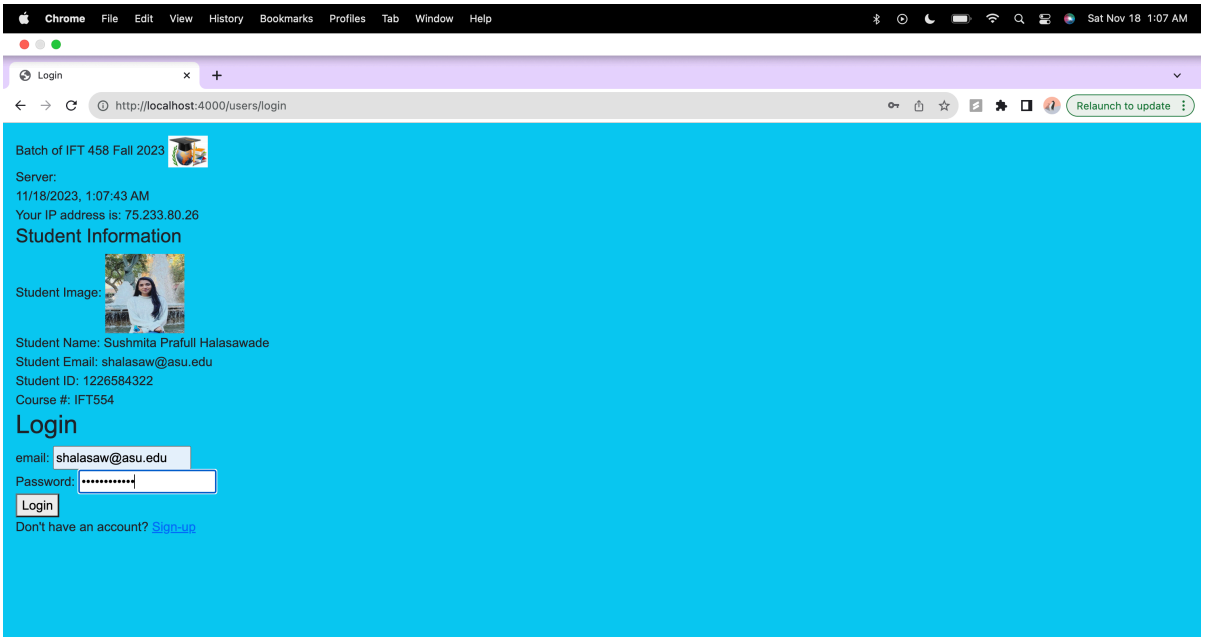
Register:



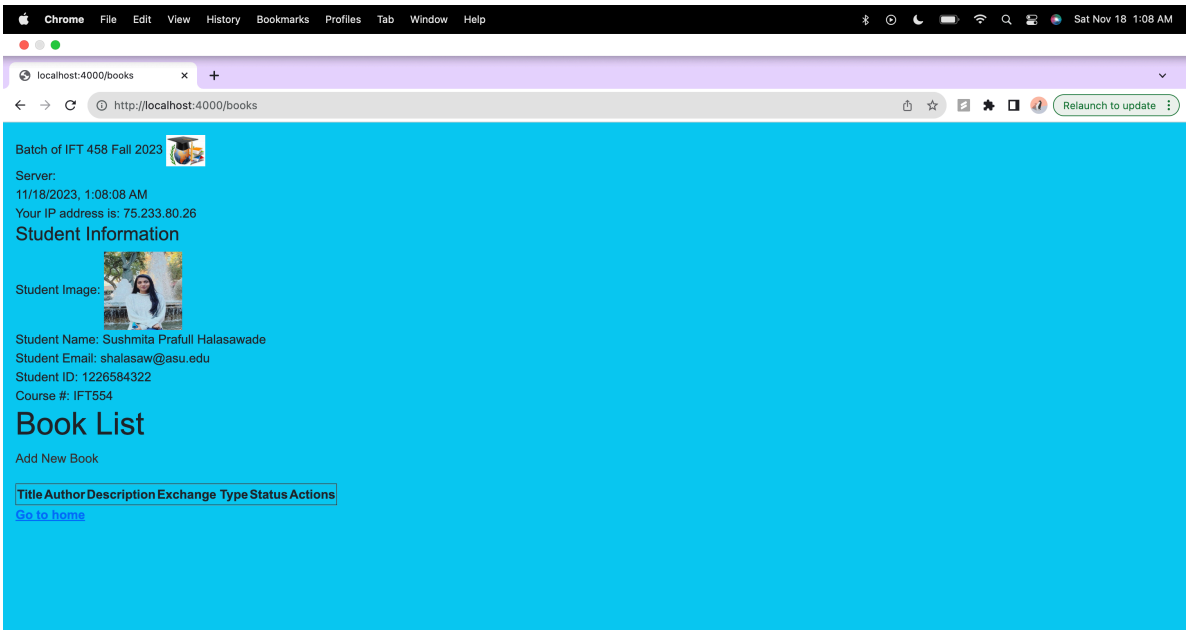
Auth success:



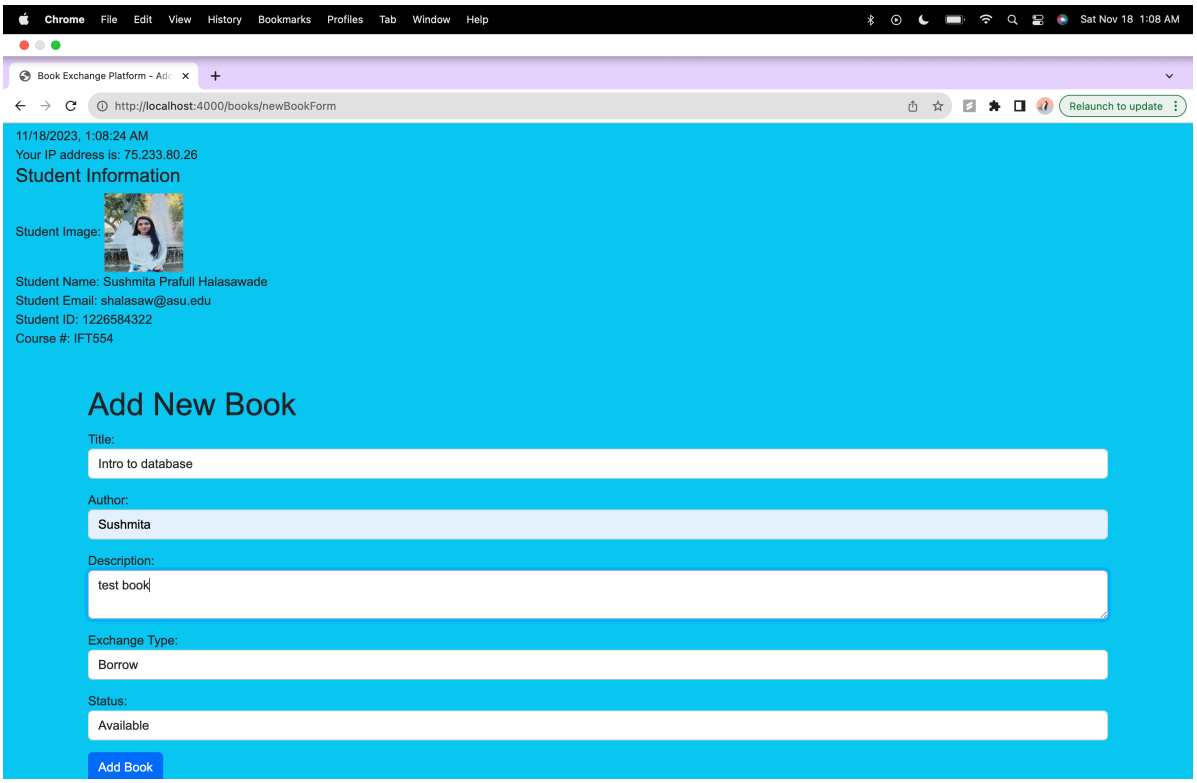
Login:



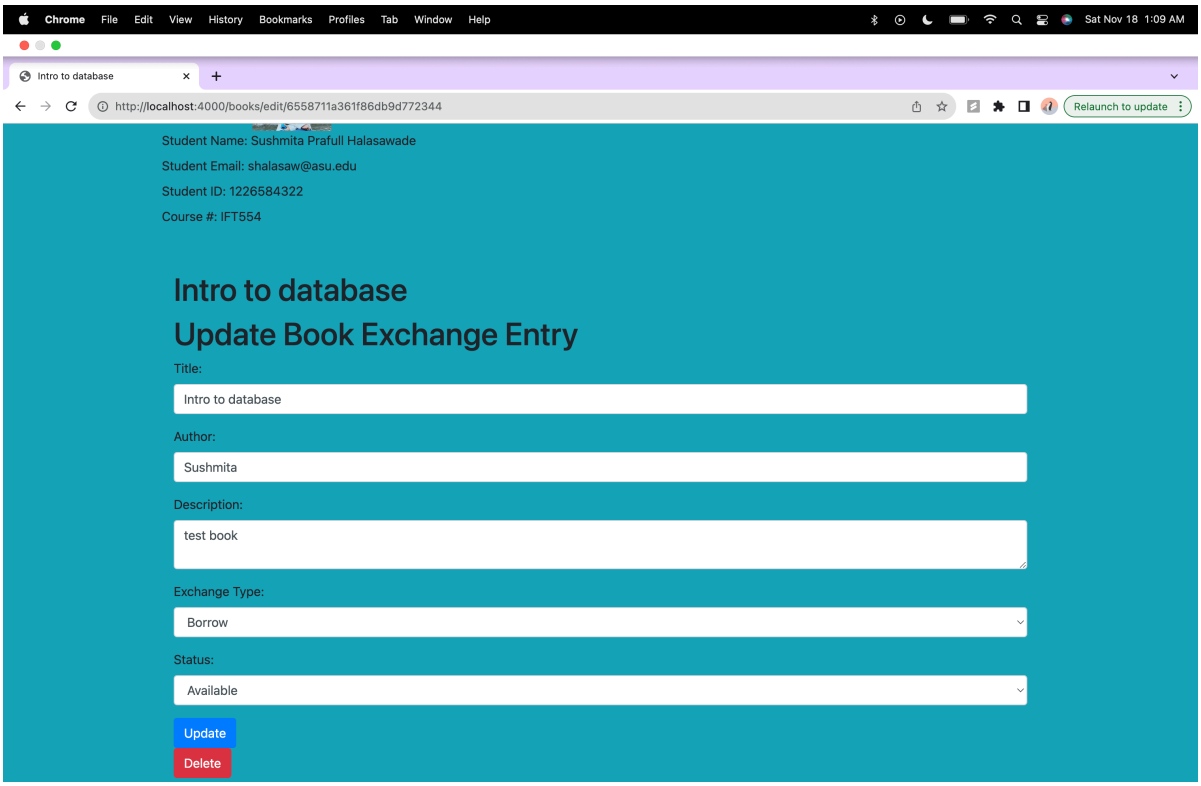
Book exchange:

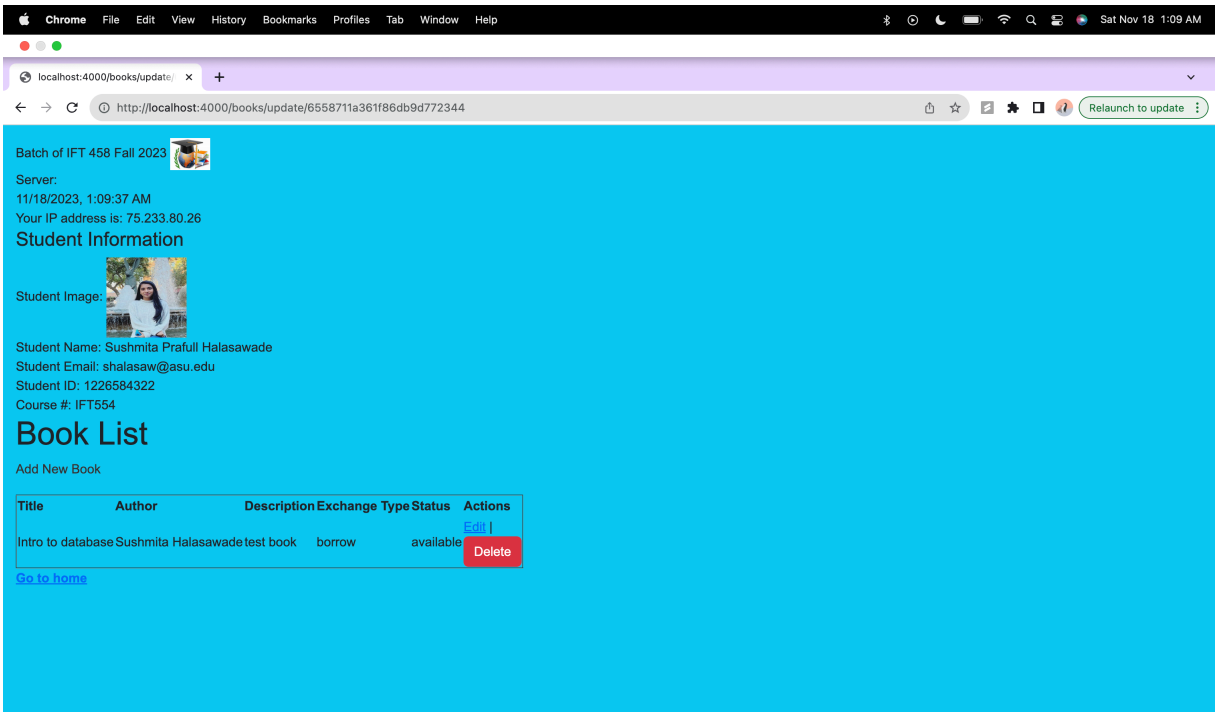


Add new book:

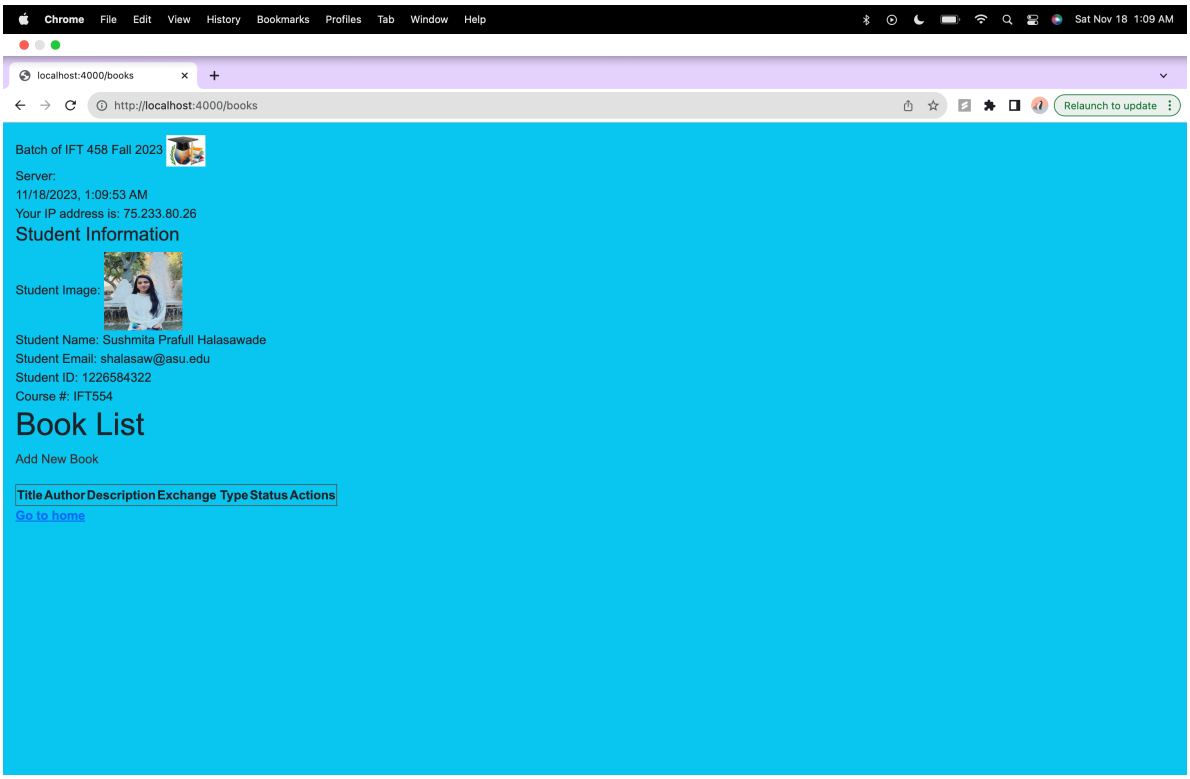


Update book:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sat Nov 18 1:09 AM
Intro to database
http://localhost:4000/books/edit/6558711a361f86db9d772344
Relaunch to update
Student Name: Sushmita Prafull Halasawade
Student Email: shalasaw@asu.edu
Student ID: 1226584322
Course #: IFT554
Intro to database
Update Book Exchange Entry
Title:
Intro to database
Author:
Sushmita
Description:
test book
Exchange Type:
Borrow
Status:
Available
Update
Delete

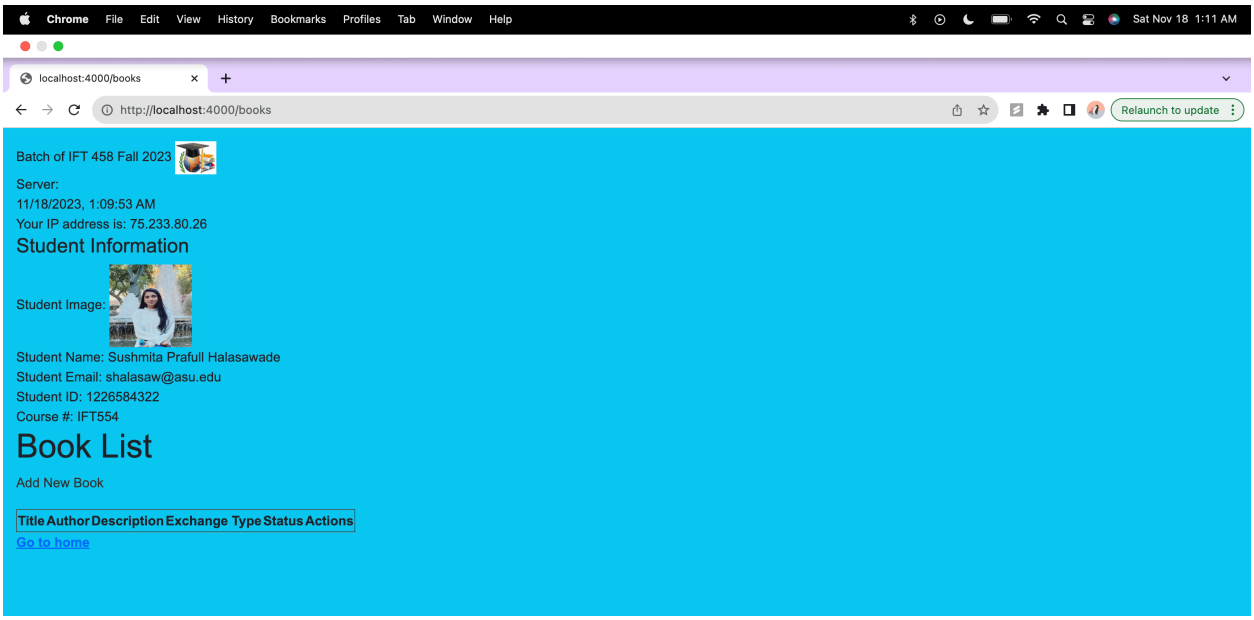

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sat Nov 18 1:09 AM
localhost:4000/books/update/
http://localhost:4000/books/update/6558711a361f86db9d772344
Relaunch to update
Batch of IFT 458 Fall 2023
Server:
11/18/2023, 1:09:37 AM
Your IP address is: 75.233.80.26
Student Information
Student Image:
Student Name: Sushmita Prafull Halasawade
Student Email: shalasaw@asu.edu
Student ID: 1226584322
Course #: IFT554
Book List
Add New Book
Title Author Description Exchange Type Status Actions
Intro to database Sushmita Halasawade test book borrow available Edit | Delete
Go to home

delete book:

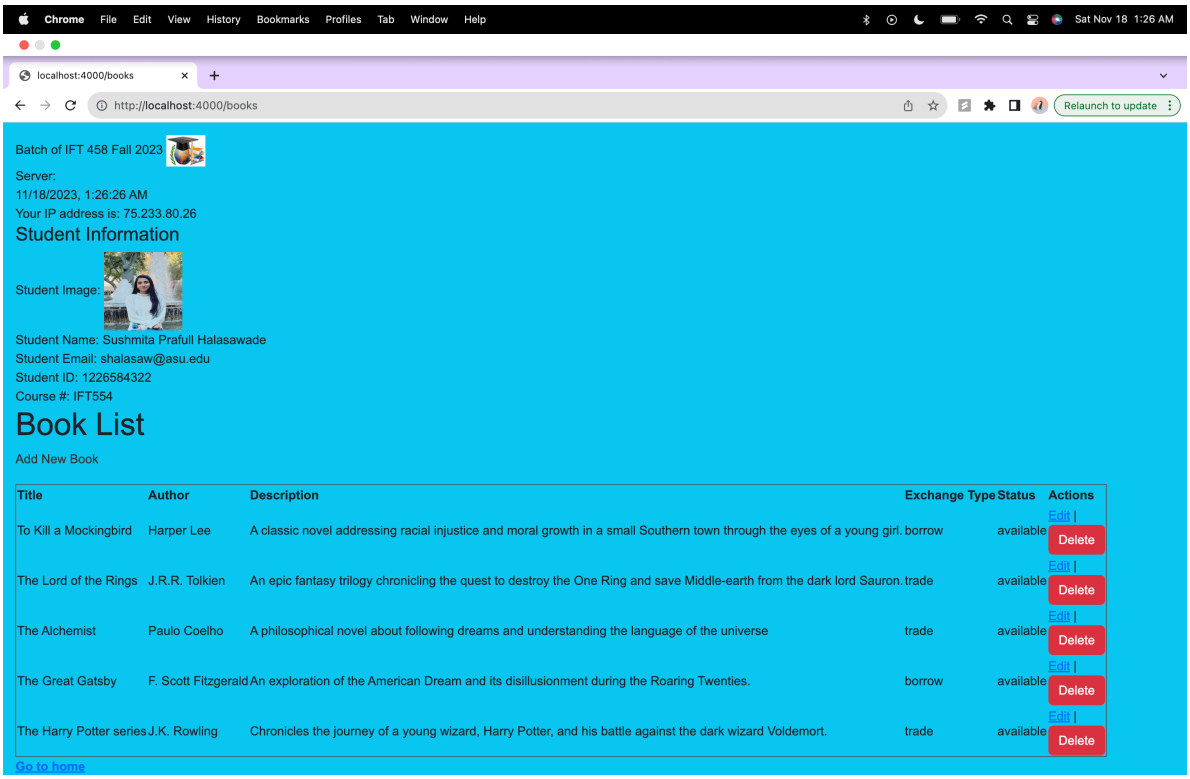


Adding 5 books in the bookexchange:

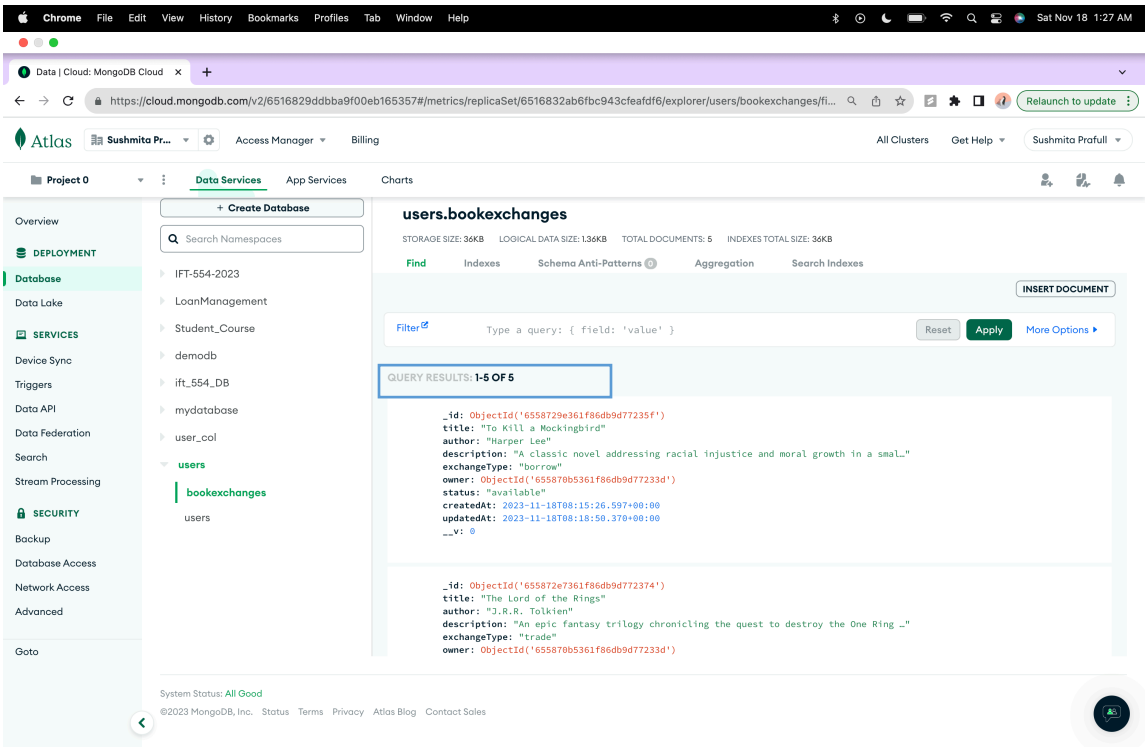
Before addition: empty book list



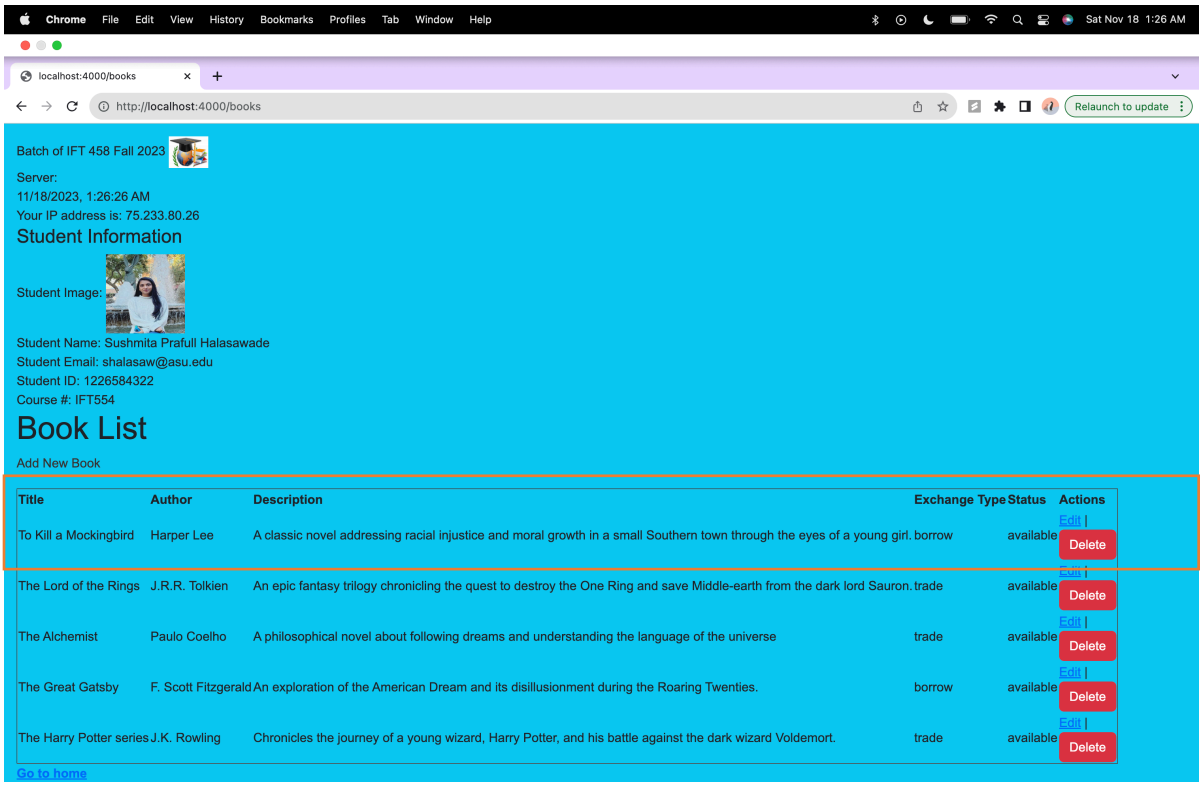
After addition of 5 books:



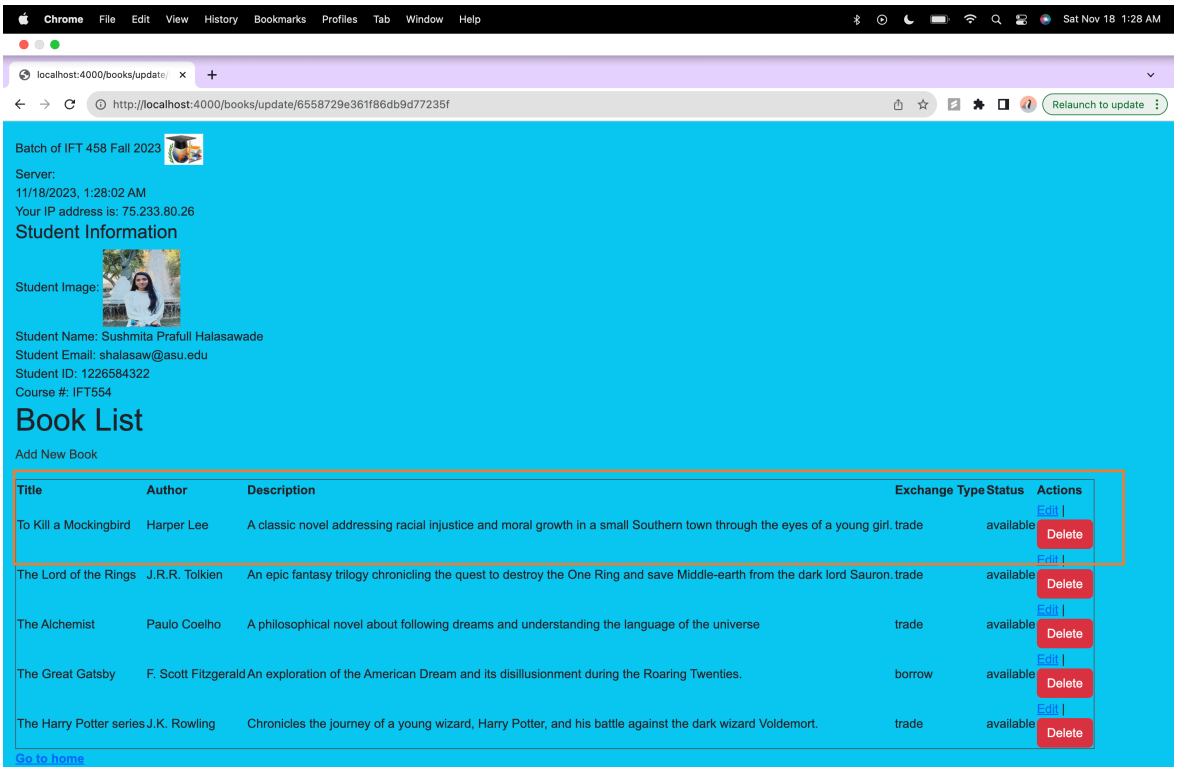
Database: 5 documents added in book exchange collection.



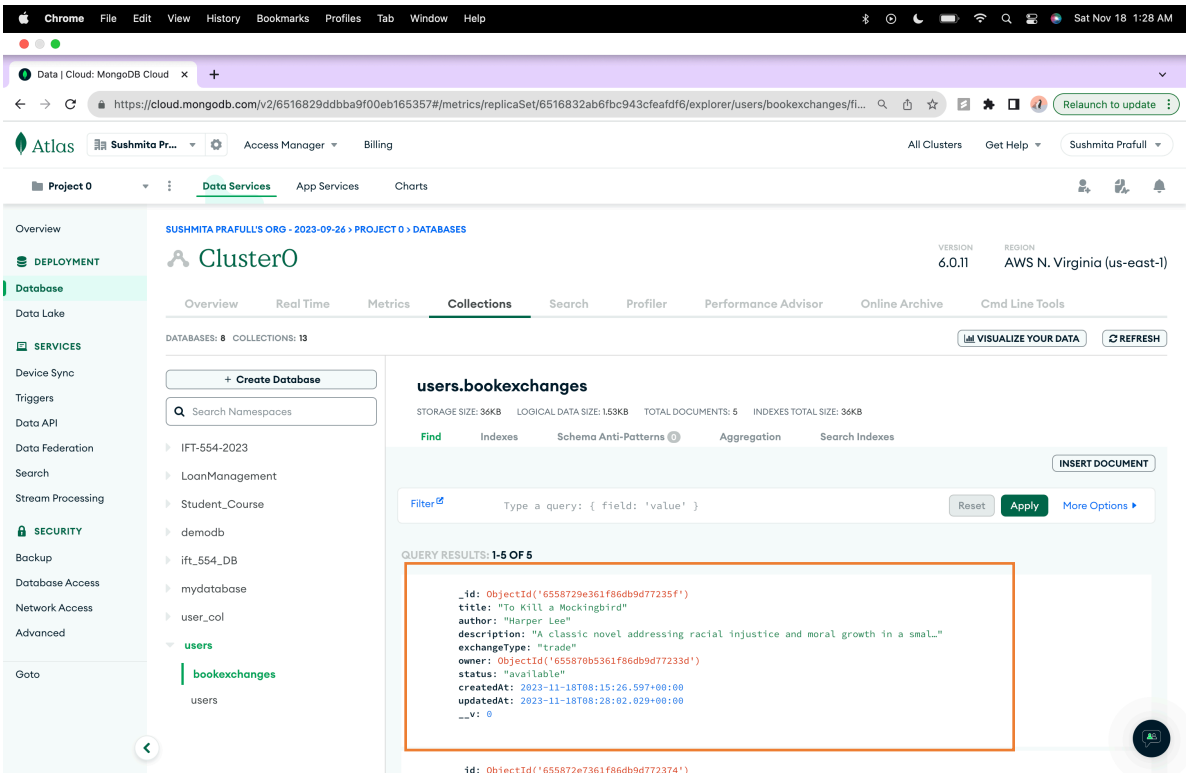
Before updating book: to kill a mockingbird:



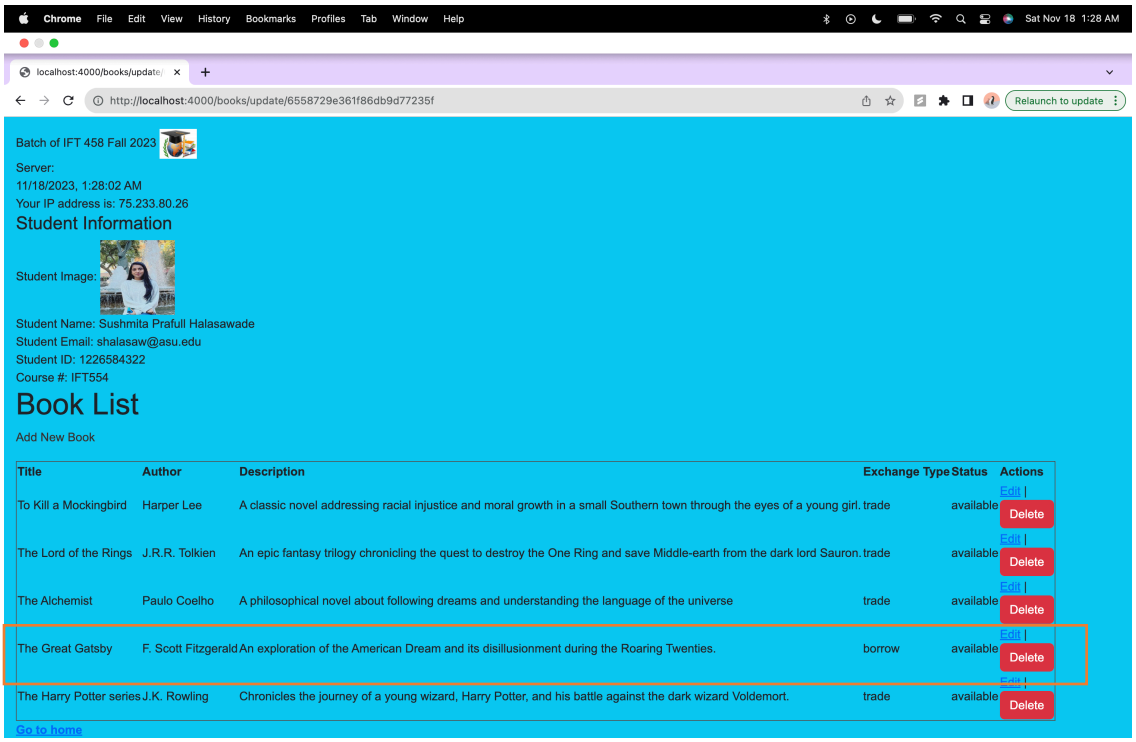
After updating the book: exchange changed from borrow to trade.



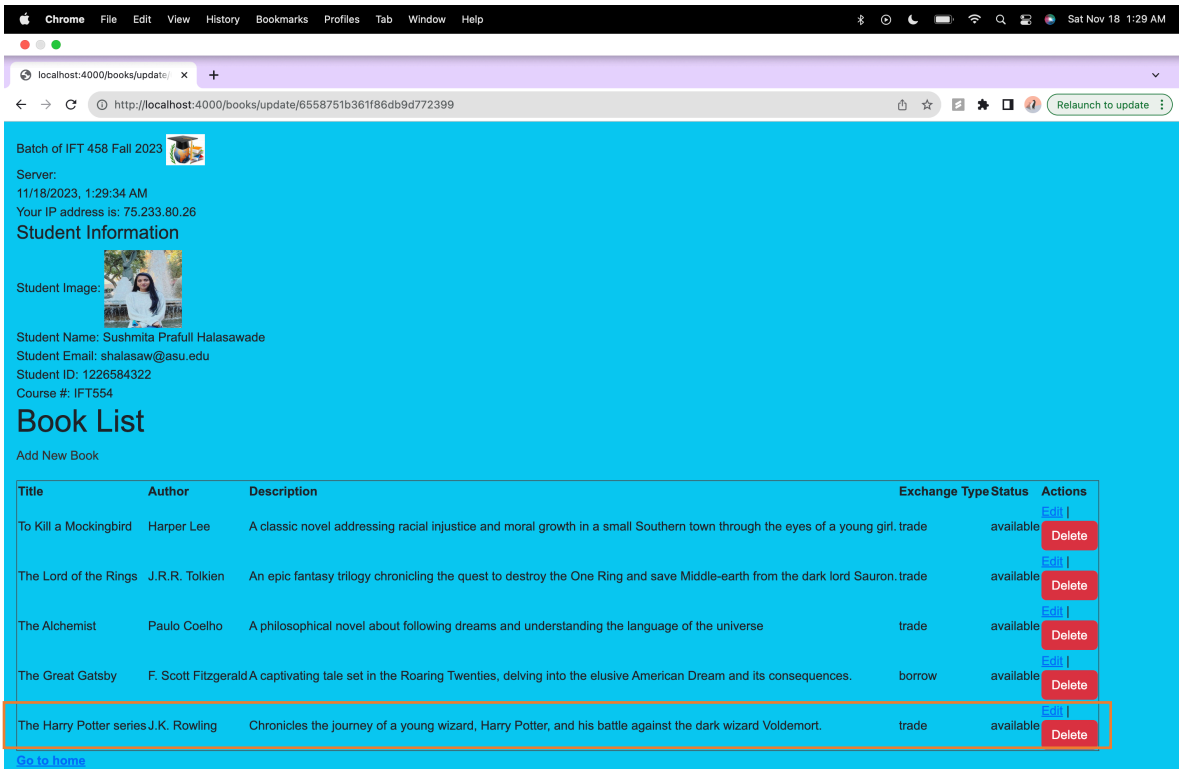
Database:



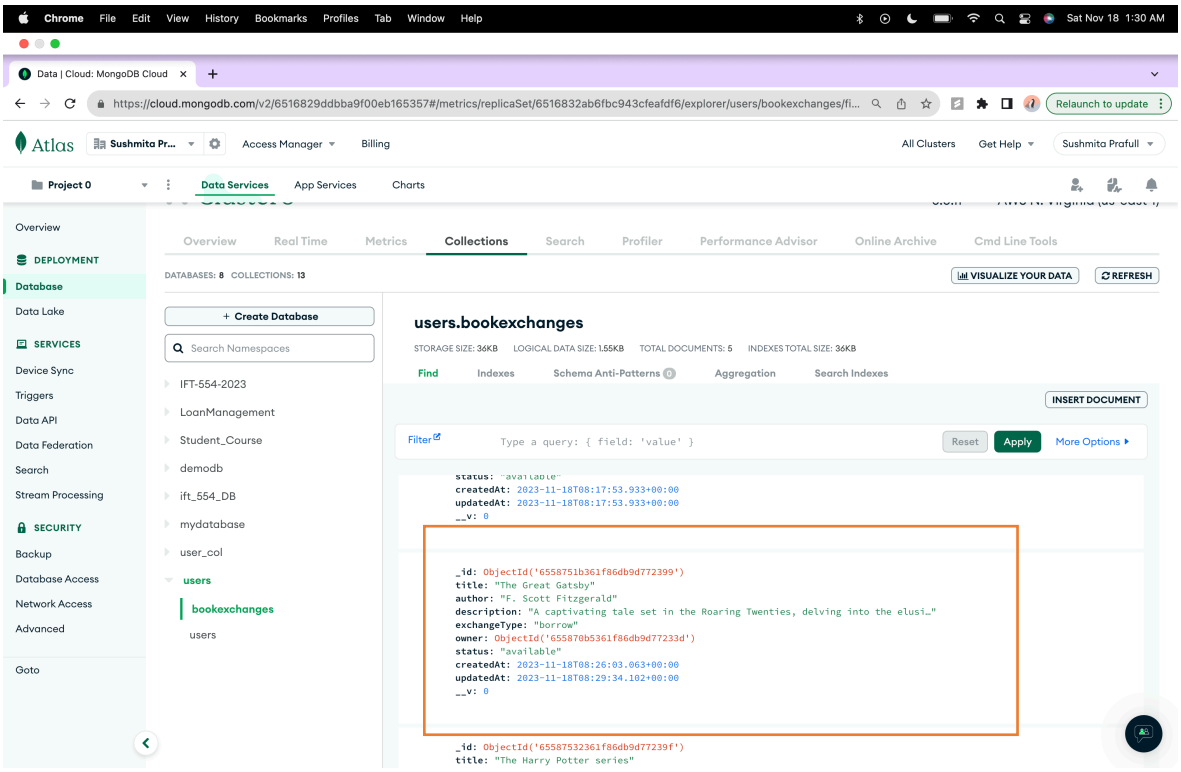
Before updating book: the great Gatsby:



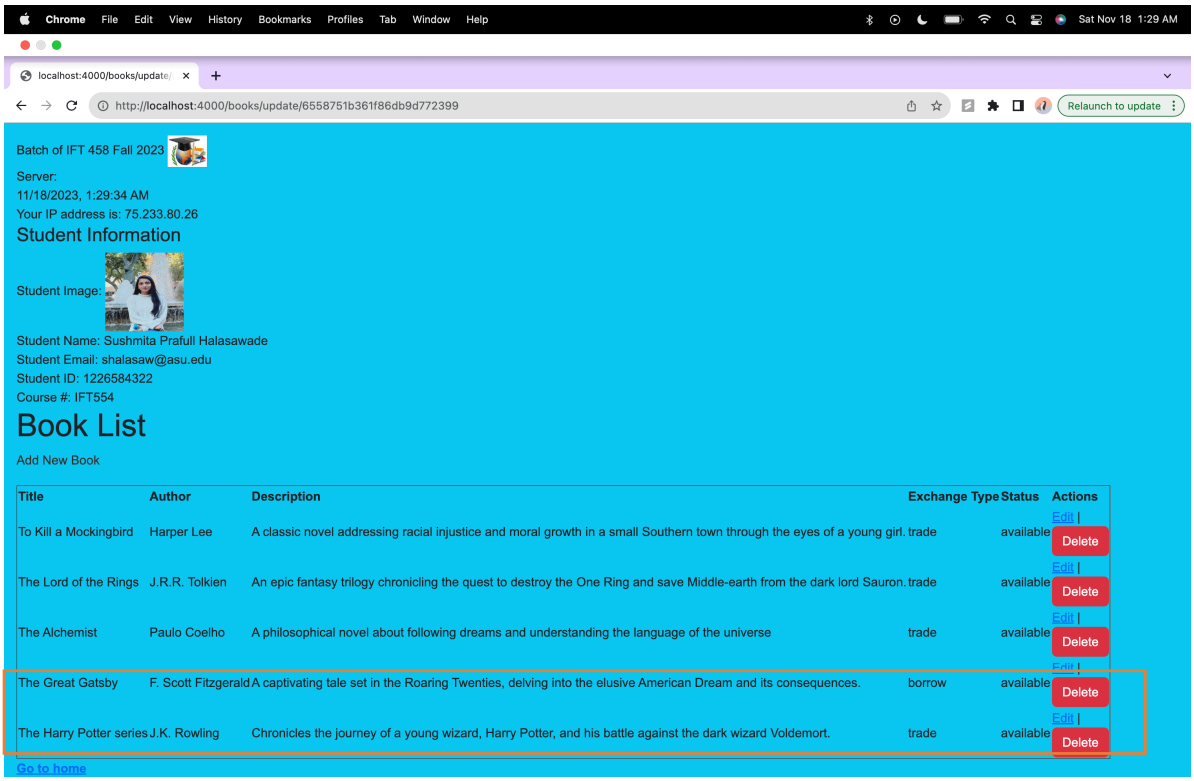
After updation of description:



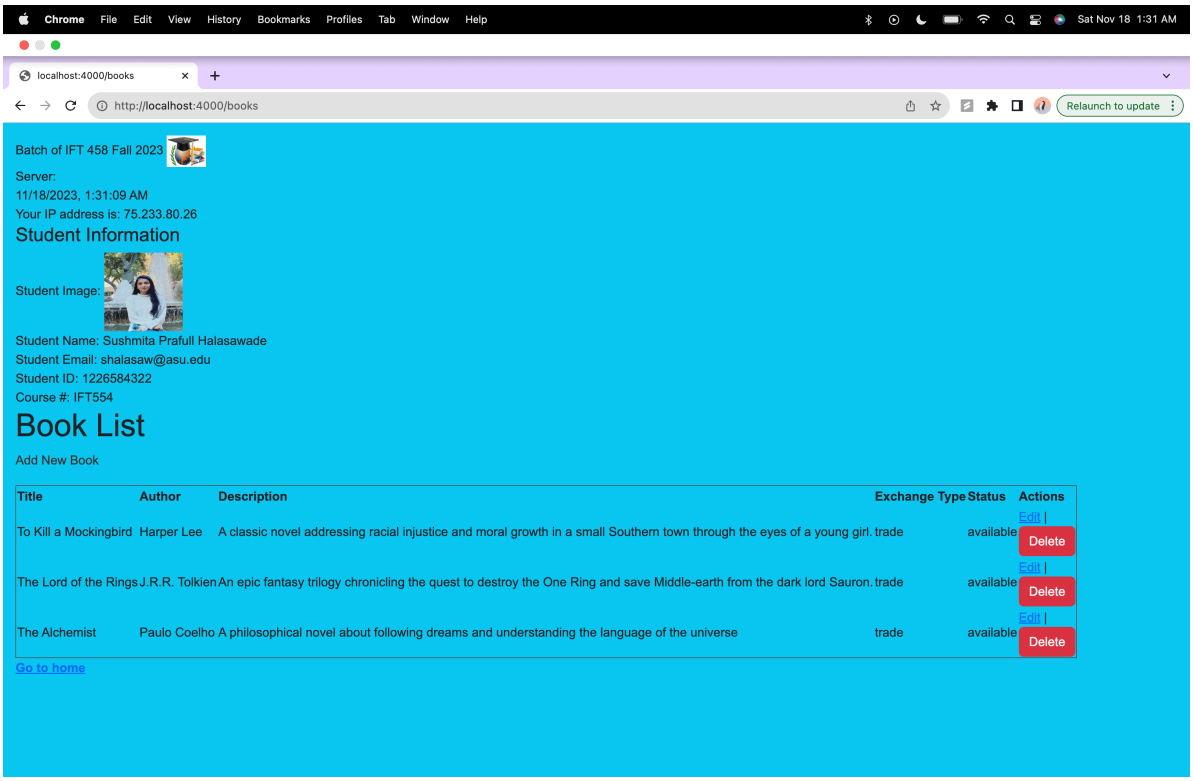
Database:



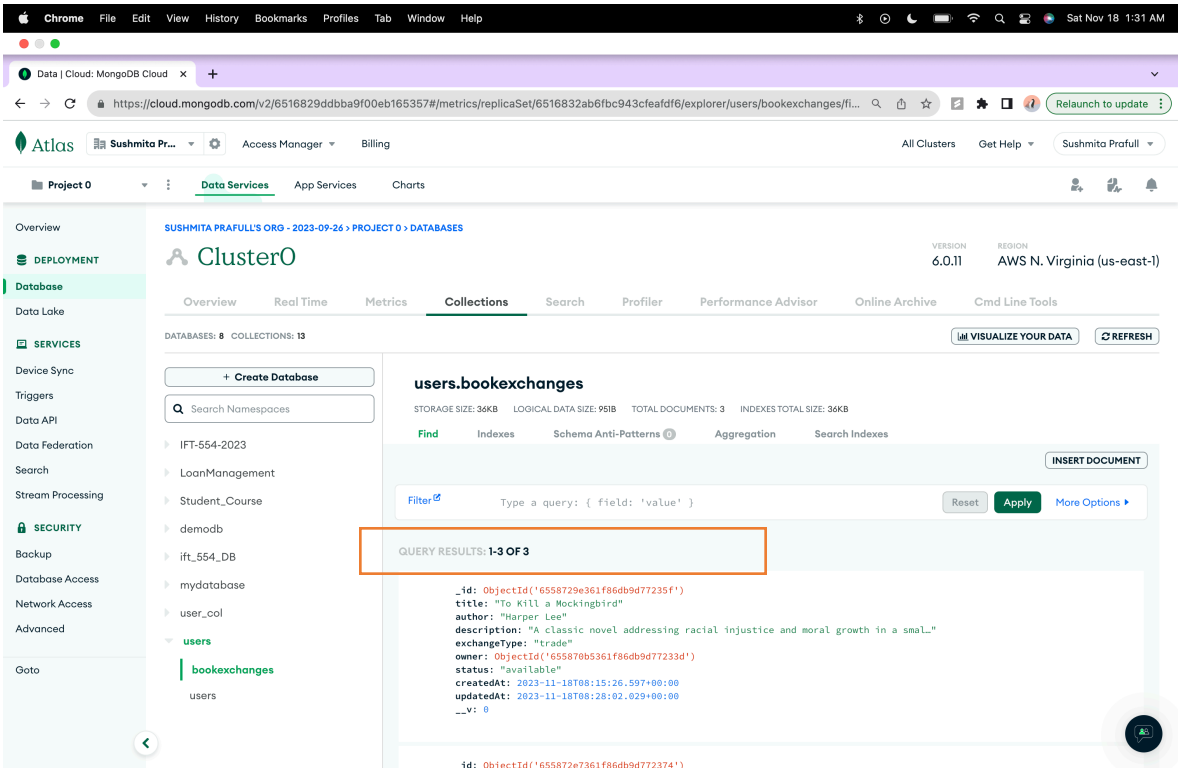
Before deleting:



After deleting book great Gatsby and harry potter series:

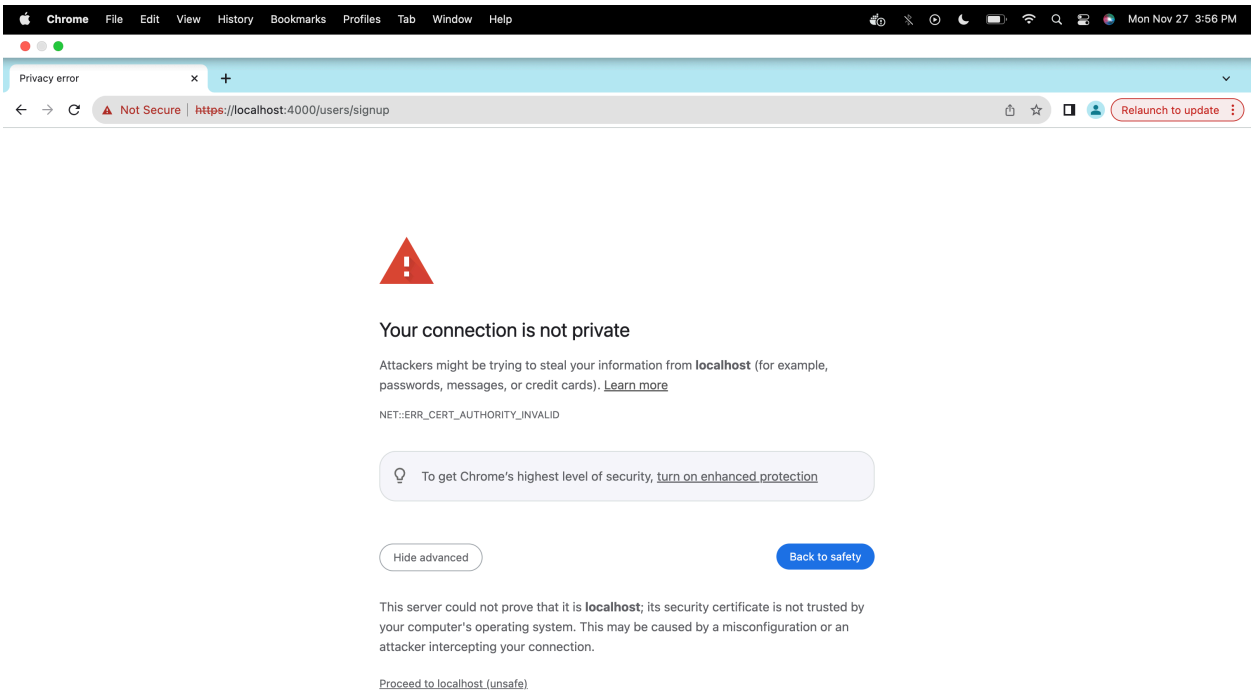


Database: 3books available in book exchange

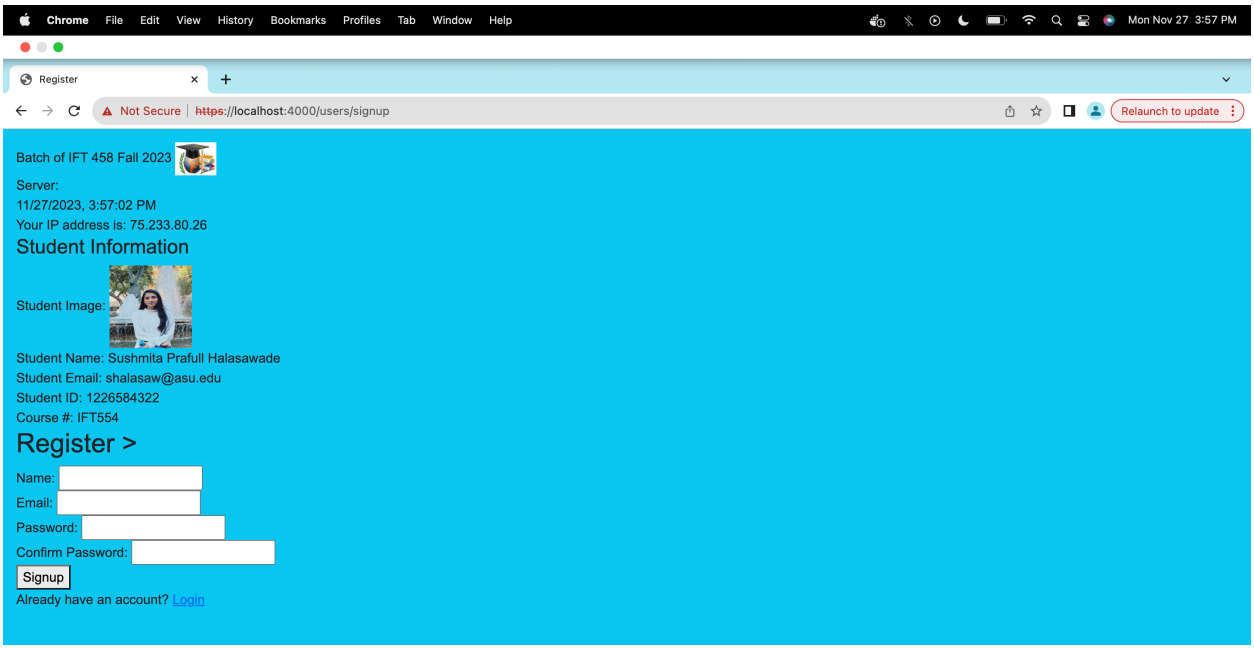


HTTPS implementation:

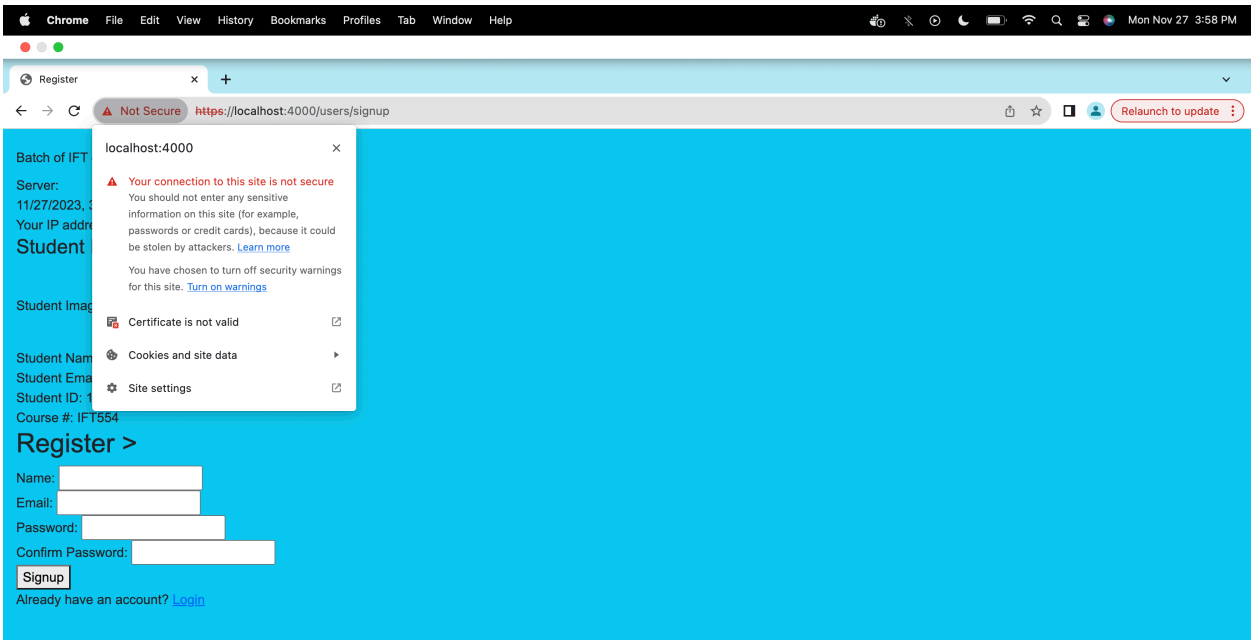
Accessing signup link: https://localhost:4000/users/signup



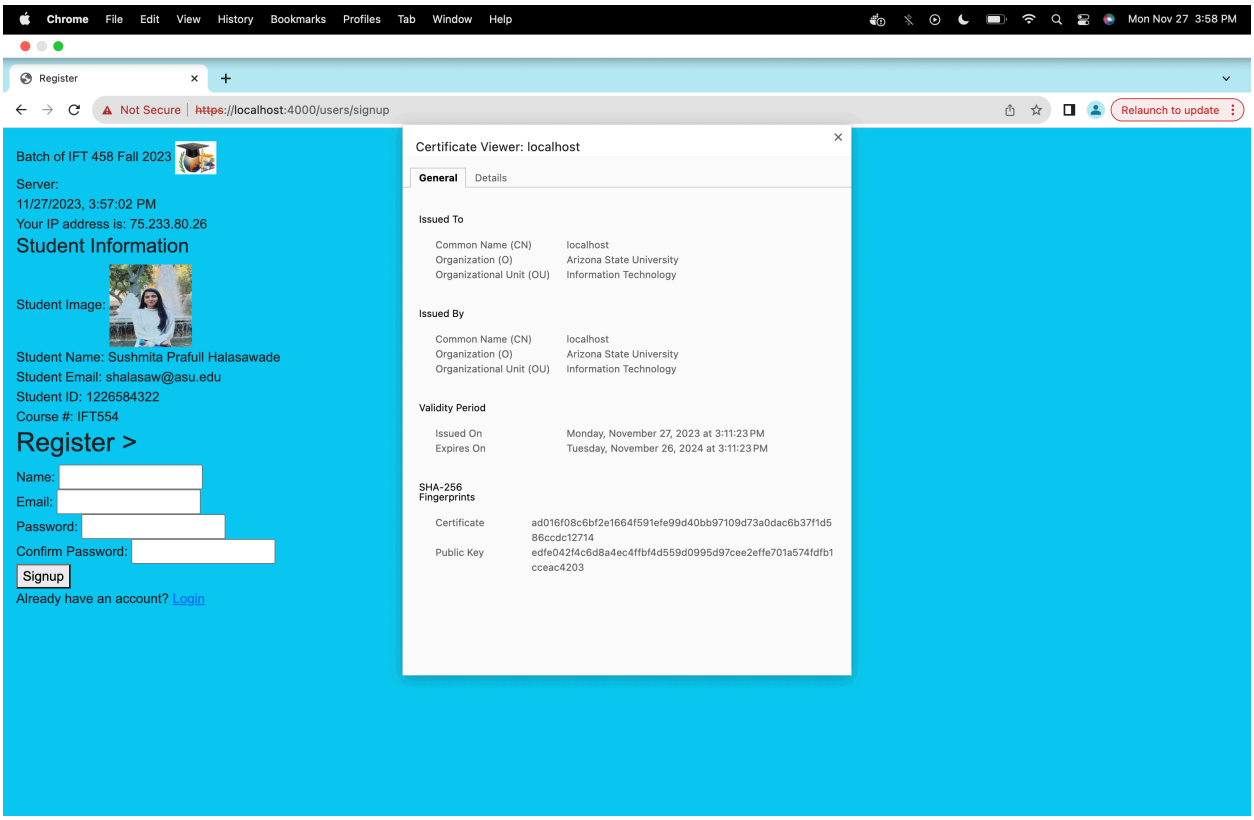
Proceed with localhost (unsafe):



Site is not secure because certificate is not valid:



Certificate and key details:



Implementation:

Challenges Faced During Implementation:

Secure User Authentication: Implementing a robust and secure user authentication system was crucial to protect user data and ensure privacy. The challenge lay in creating an authentication process that was both secure against threats and user-friendly.

Efficient Data Management: Transitioning from a basic array-based system to a more scalable MongoDB database posed challenges. The primary concern was efficiently managing a growing database of books and users, ensuring fast and reliable access to data.

Switching to HTTPS from HTTP: The transition to HTTPS was essential for enhancing security, particularly for protecting data during transmission. This required setting up SSL/TLS certificates and ensuring compatibility across all parts of the application.

Workarounds and Solutions Applied:

For Secure User Authentication: JWT (JSON Web Tokens) and bcryptjs were used for authentication and password hashing, respectively. These technologies provided a balance between security and performance, offering a reliable way to handle user sessions and protect sensitive data.

For Efficient Data Management: MongoDB, a NoSQL database, was integrated to handle the expanding data. It offered flexibility, scalability, and efficient handling of large volumes of unstructured data, making it a suitable choice for a dynamic platform like the Book Exchange website.

For HTTPS Implementation: The server was configured to handle HTTPS requests, ensuring all data transferred between the server and client is encrypted. This was vital for maintaining data integrity and confidentiality, especially for login credentials and personal user information.

Key Functionalities and Features:

User Registration and Authentication: Users can sign up and log in, with their credentials securely managed and authenticated.

Book Management: Users can add, update, delete, and browse books. This feature includes detailed information about each book, such as title, author, and exchange type.

Book Exchange Process: The platform facilitates the exchange of books between users, including the ability to request an exchange and view available books.

Security Features: Alongside secure authentication, the website includes middleware for route protection, ensuring only authenticated users can access certain functionalities.

Learning Outcomes:

The development of the "Book Exchange" website provided an immersive learning experience, solidifying my understanding of several key concepts from Modules 5 to 8 and enhancing both my technical skills and broader comprehension of web development.

Technically, I gained in-depth knowledge of authentication and authorization mechanisms, particularly mastering the use of JSON Web Tokens (JWT) and OAuth. Implementing JWT for secure user sessions and OAuth for streamlined third-party authentications offered valuable hands-on experience, emphasizing the importance of robust security in web applications. Transitioning the website to HTTPS using mTLS (Mutual Transport Layer Security) was another crucial learning curve, not only heightening my awareness of data security during transmission but also sharpening my skills in configuring and securing web servers.

The use of EJS for view designs introduced me to the dynamics of server-side rendering, blending JavaScript with HTML for dynamic content generation, significantly improving my design capabilities. Additionally, learning to test APIs with a REST client and utilizing RESTful APIs for efficient server-client communication were pivotal in understanding back-end to front-end interactions. The deployment phase using Docker offered insights into containerization and its benefits in ensuring consistency across different environments.

Navigating through these modules, I developed a keen sense of problem-solving and adaptability. Implementing JWT and OAuth, for instance, required not just coding skills, but an understanding of user behavior and security needs. Working with HTTPS and mTLS deepened my commitment to user privacy and data security. Designing views with EJS and testing APIs cultivated my attention to detail and the importance of user experience. Finally, deploying with Docker highlighted the significance of smooth and reliable application performance.This project was more than just coding; it was about creating a secure, user-friendly, and efficient platform that resonates with the needs of its users.

Constructive Feedback:

- Modern Front-End Frameworks: The course mainly taught backend technology and how to render websites server-side with EJS. Including lessons on popular front-end frameworks like React, Angular, or Vue.js would give a more complete understanding of both front-end and back-end development, which is what full-stack developers are expected to know.

- Real-World Case Studies and Examples: Adding more examples and case studies from actual projects can help students understand how these technologies are used in real situations. This makes it easier to see how the theory they learn fits into practical work.
- Cloud Services and Scalability: Since cloud-based services are becoming more important, a section on cloud computing and how to manage big-scale applications would be really useful. This could involve learning about major cloud providers like AWS, Azure, or Google Cloud, and how to use Kubernetes for managing multiple application containers.
- In-Depth Backend Development: Expanding the parts of the course that deal with backend development would be useful. This should include more complex topics like how to manage large databases, building applications with many separate services (microservices), and making sure the server side of applications runs efficiently.

References:

JWT.IO - JSON Web Tokens Introduction. (n.d.). Jwt.io. https://jwt.io/introduction

Mongoose. (2011). *Mongoose ODM v5.8.2*. Mongoosejs.com. https://mongoosejs.com/

ejs. (2018, May 5). Npm. https://www.npmjs.com/package/ejs

oAuth. (2020). *OAuth 2.0 — OAuth*. Oauth.net. https://oauth.net/2/

Porter, B. (2021, February 7). *What is mTLS and How Does it Work?* Medium. https://freedomben.medium.com/what-is-mtls-and-how-does-it-work-9dcdbf6c1e41