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

Naga Sumanth Vema

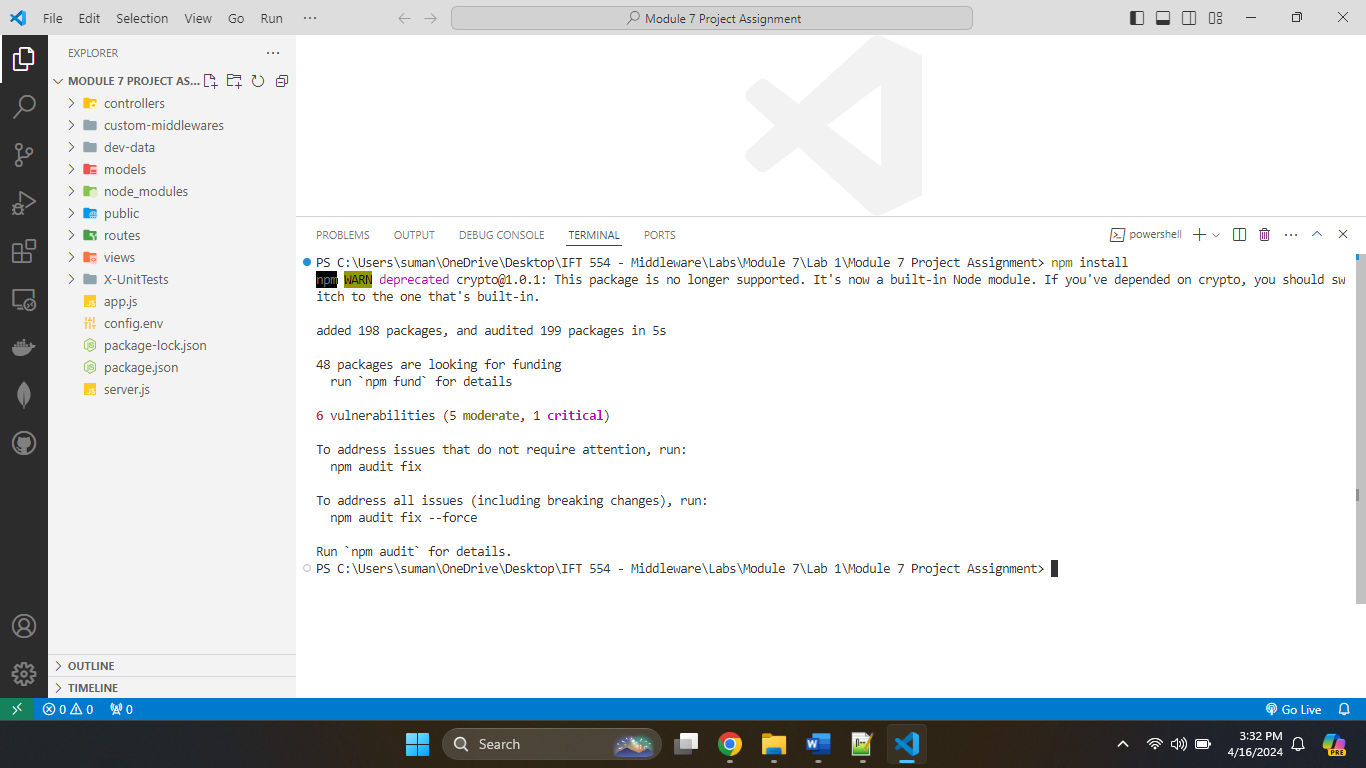
IFT 554: Middleware Programming & Database Security

Dinesh Sthapit

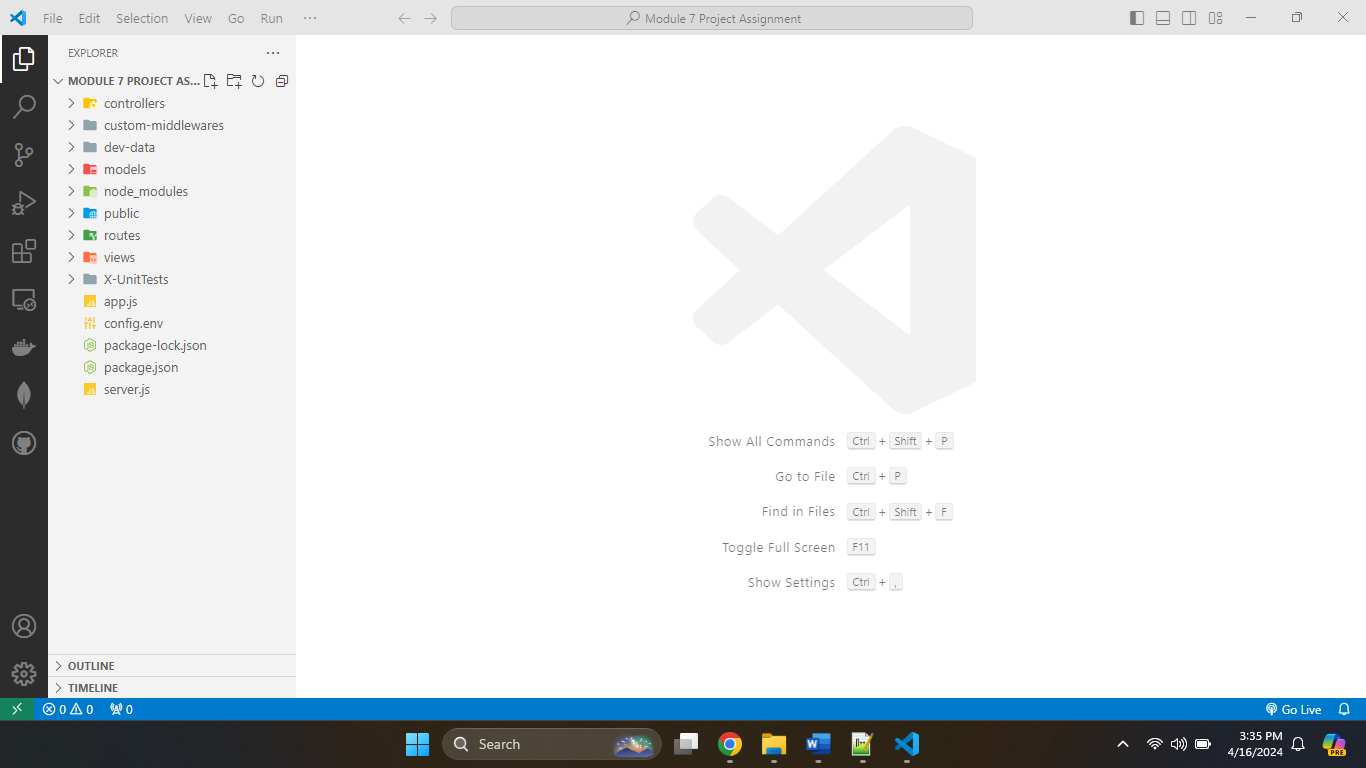
April 21st, 2024

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

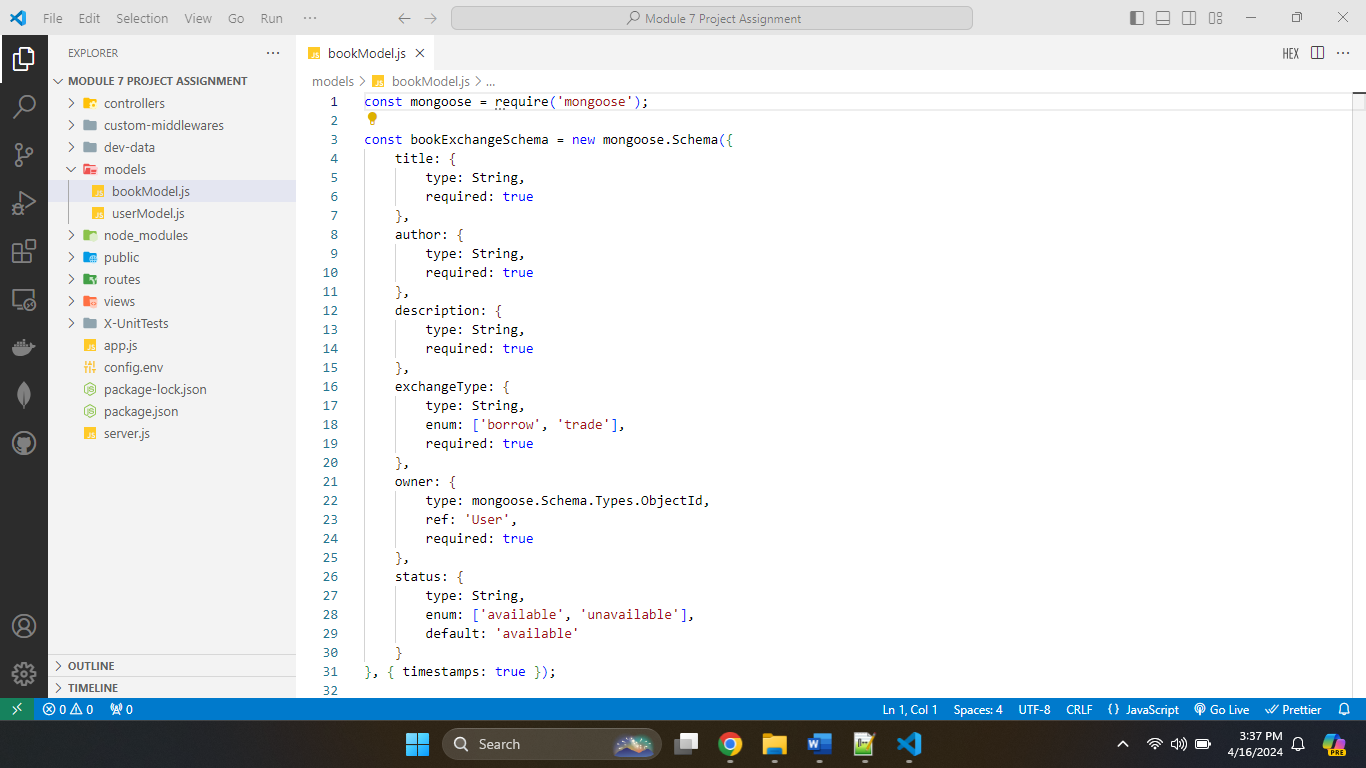
Screenshot of the terminal showing successful installation of dependencies.



Screenshot of the project directory structure showcasing MVC organization.



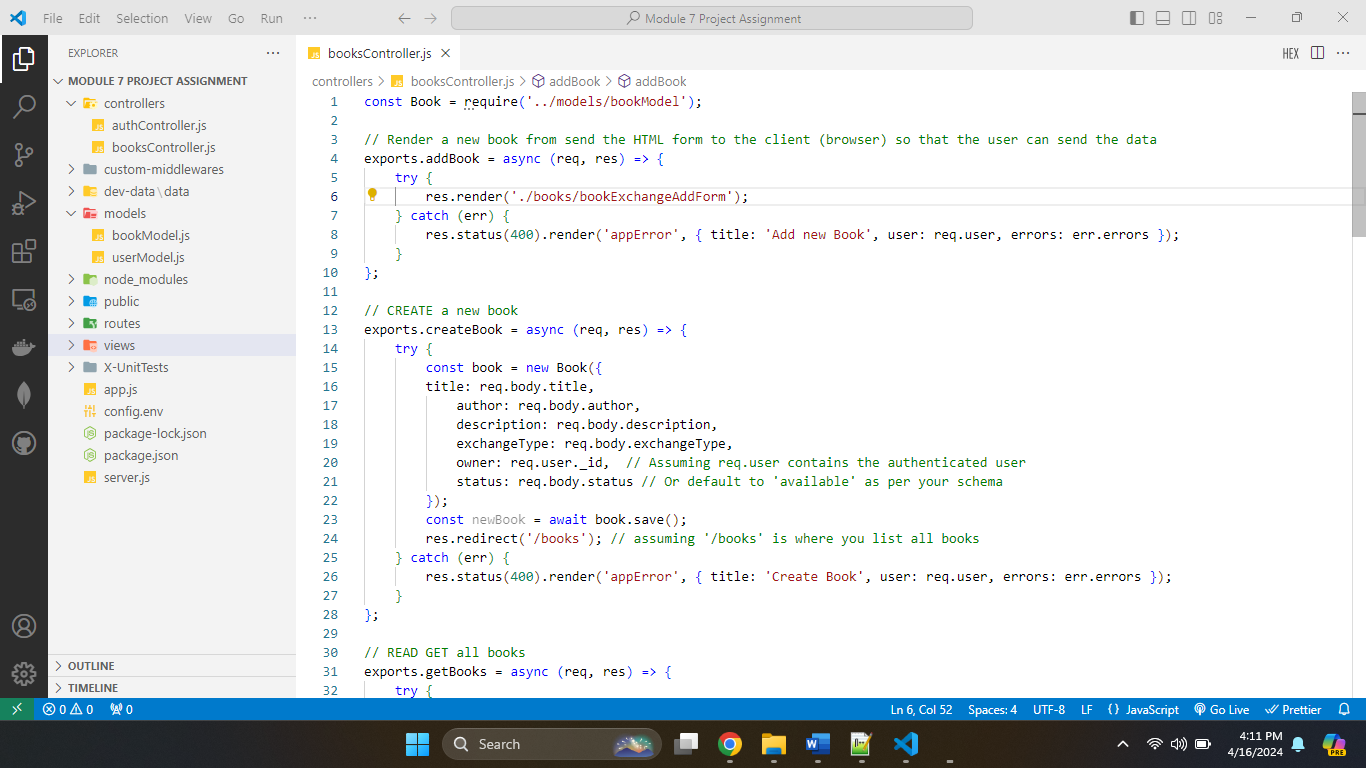
**Screenshot of the BookModel.js file and a brief explanation in your document.**

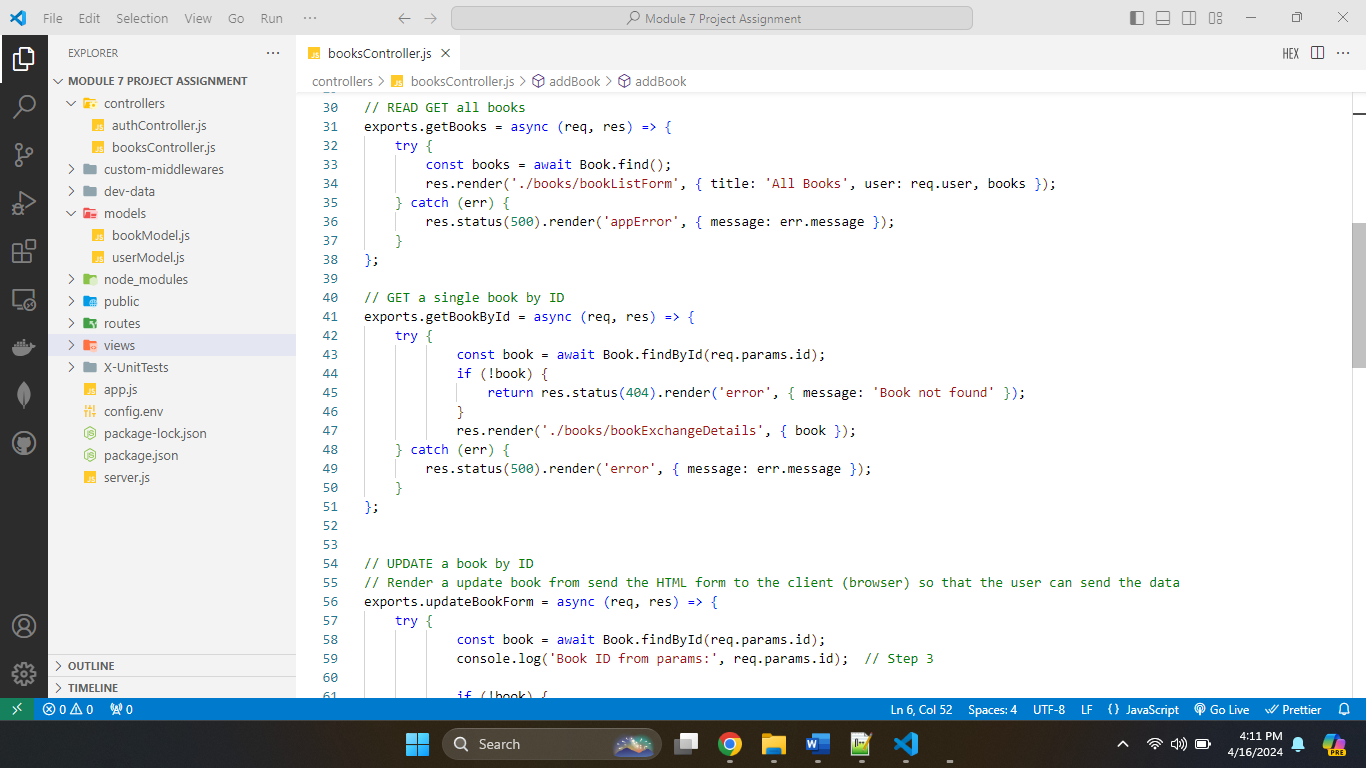


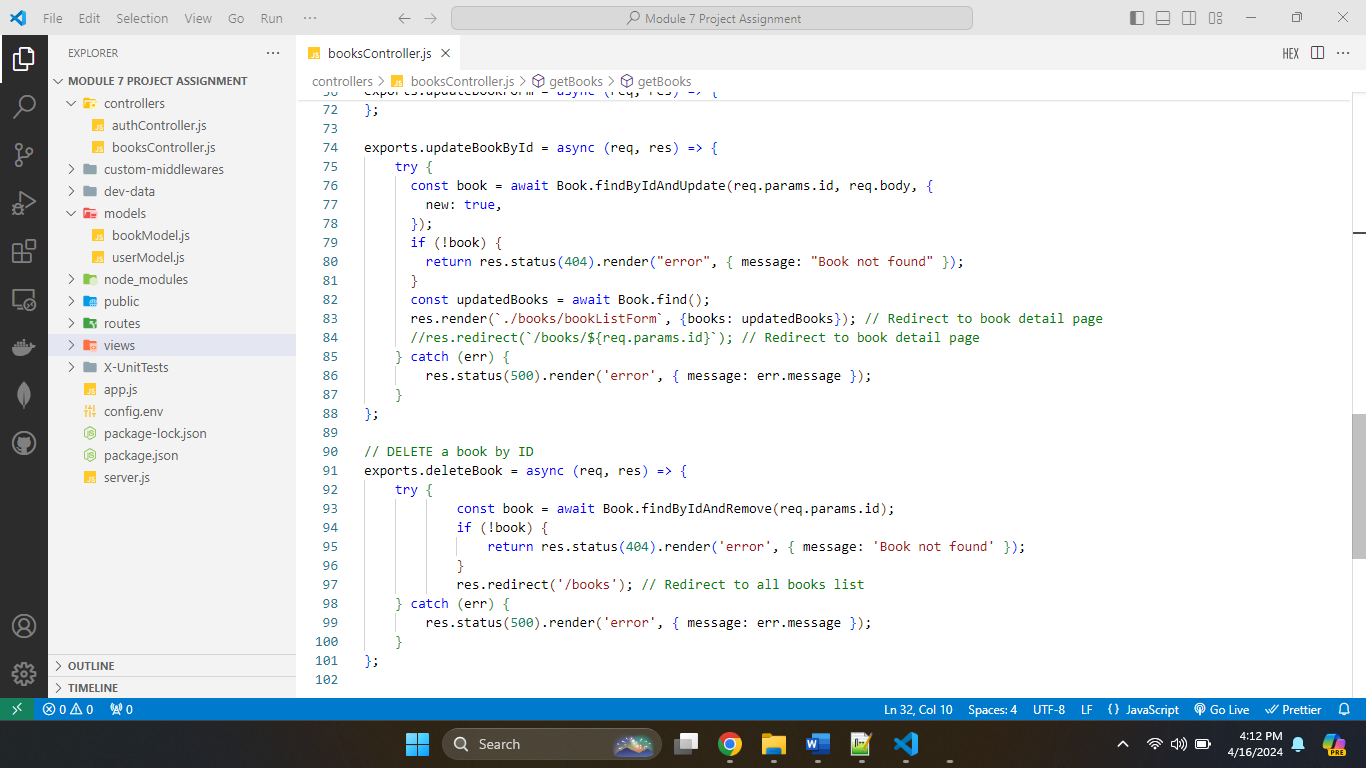
The bookModel.js file specifies the structure for managing book data in the book exchange application. Using the mongoose library, the schema defines the many properties that each book document will have. The properties include title, author, description, exchangeType, owner, and status. Each field is precisely described, including its data type, such as String, as well as any special constraints, such as needed. Furthermore, some data, such as exchangeType and status, are limited by enumerations to ensure consistency and correctness. The owner field is extremely crucial, as it refers to the user who owns the book. This reference is made using the ref attribute, which links it to the 'User' model.

The schema includes default values and timestamps, such as 'available' for the status field and {timestamps: true } for automatically adding createdAt and updatedAt fields. Finally, the schema is turned to a Mongoose model named 'BookExchange' using mongoose.model, allowing for smooth interaction with the MongoDB collection. This model contains the functionality needed to manage book data and is exported for usage across the application.

**Screenshot of the bookController.js with the implemented logic.**







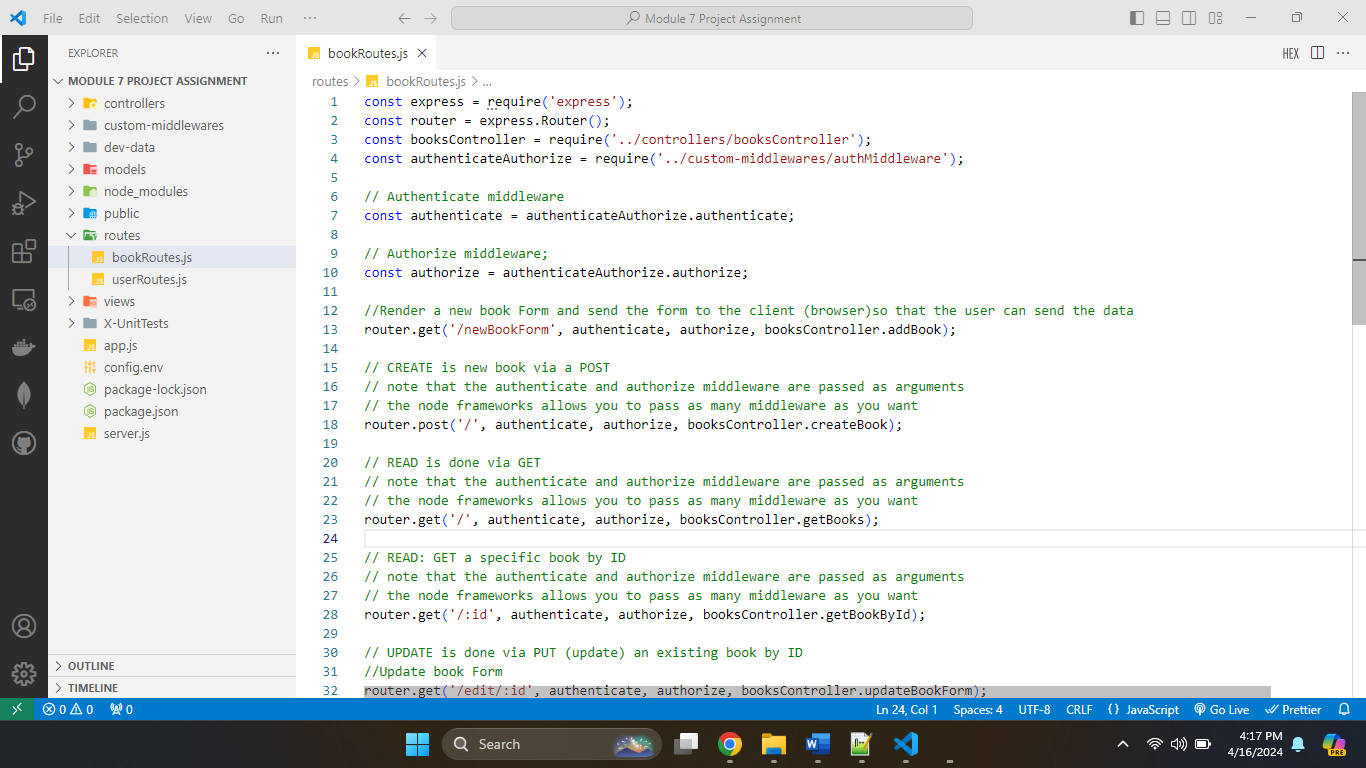
The bookController.js file encapsulates the logic for controlling book-related actions in the book exchange application. First, the addBook function displays a form where users may enter new book information. Any rendering issues result in a status 400 response and an appropriate error notice. The createBook function then analyzes the form data and creates a new book instance. After successful creation, it saves the book to the database and brings the user to the book list page. Any mistakes throughout this procedure result in a status 400 response with an error message.

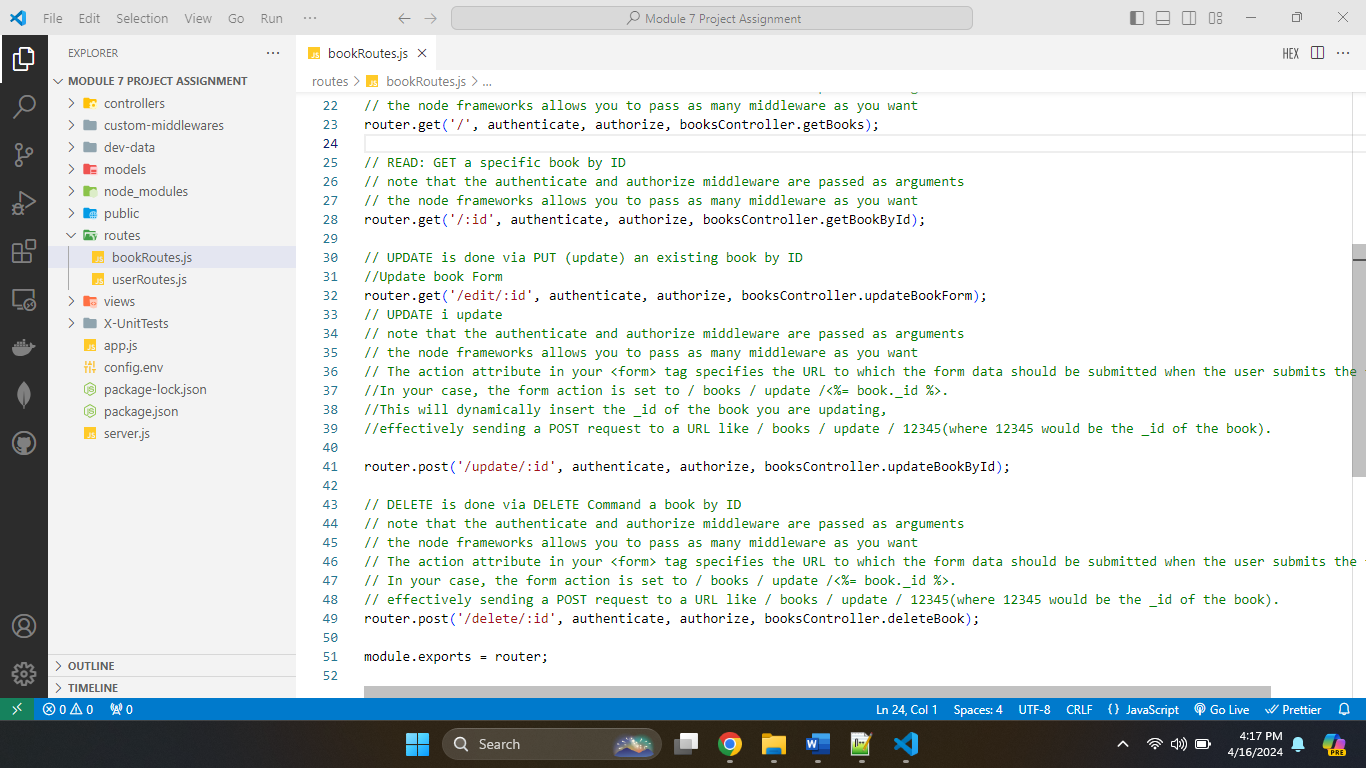
The getBooks method obtains all books from the database and displays a page with a book list. In the event of a retrieval error, it returns a 500 status code and an error message. Similarly, the getBookById method fetches a specific book using its ID and displays its data. If the requested book is not found, it gives a 404 error. Retrieval failures provide a status 500 response.

The updateBookForm method generates a form for users to update a book's information, which is pre-populated with the current data. If the requested book is not found, it gives a 404 error. Any mistakes throughout this procedure result in a status 400 response with an error message. The updateBookById method modifies a book's information depending on its ID and saves the changes to the database. It then takes the customer to the revised book information page. If the book cannot be located, it produces a status 404 error, while any update failures result in a status 500 response.

Finally, the deleteBook method removes a specific book from the database using its ID. If the book is not discovered, it gives a 404 error. It leads the user to the book list page after a successful deletion. Any deletion problems provide a status 500 response with an error notice.

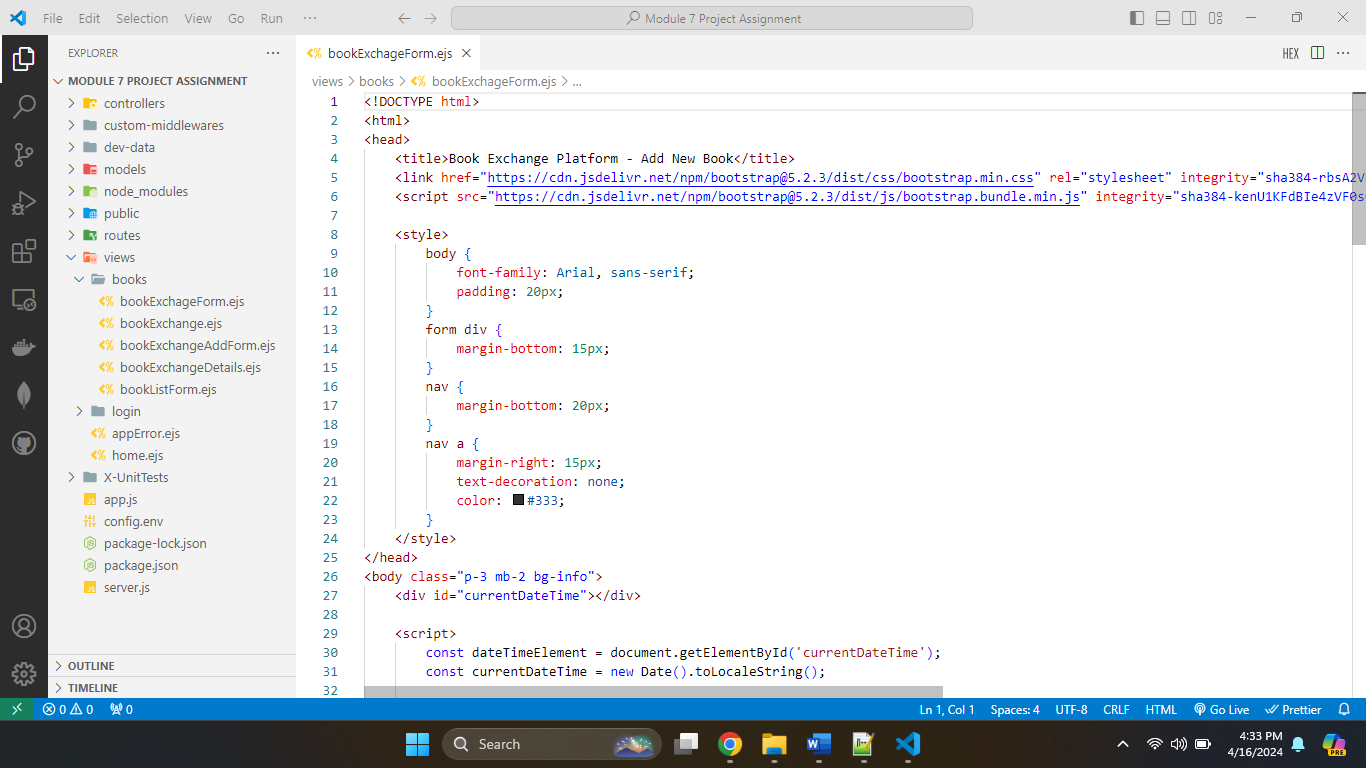
**Screenshot of the bookRoutes.js file with the defined routes.**





**Screenshots of the updated view files (\*.ejs) and the changes made.**

bookExchageForm.ejs



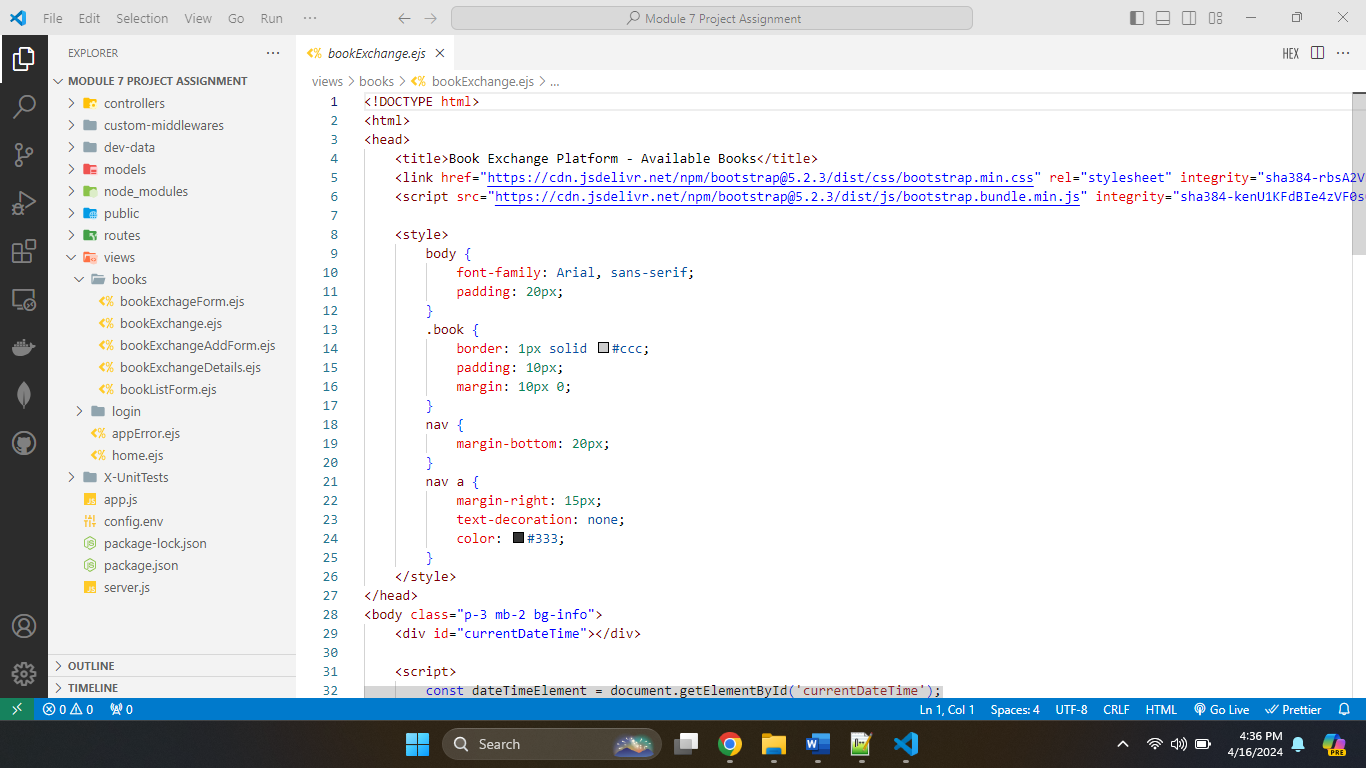
bookExchageForm.ejs is a web page that adds a new book to a book exchange platform. It starts with the standard document type declaration and HTML structure. The <head> section provides metadata and external resources, such as a title, Bootstrap CSS, and JavaScript files. Custom styles are also applied to components such as the body, form, and navigation.

The <body> section includes JavaScript code that dynamically displays the current date and time and retrieves the user's IP address via an API request. If the user is signed in, the header section shows a welcome, as well as links to the dashboard and logout. If you are not logged in, it will give buttons to signup or login.

The primary material is housed within a <section> element, which includes a form for adding new books. The form has sections for the book's title, author, genre, and description, as well as an optional file input for the cover. Upon submission, the form data is delivered to the /books/add endpoint using the POST method.

The <footer> section includes copyright information for the book sharing platform. Overall, the HTML page acts as a user interface for users to enter book information and submit it for inclusion on the site.

**bookExchange.ejs**

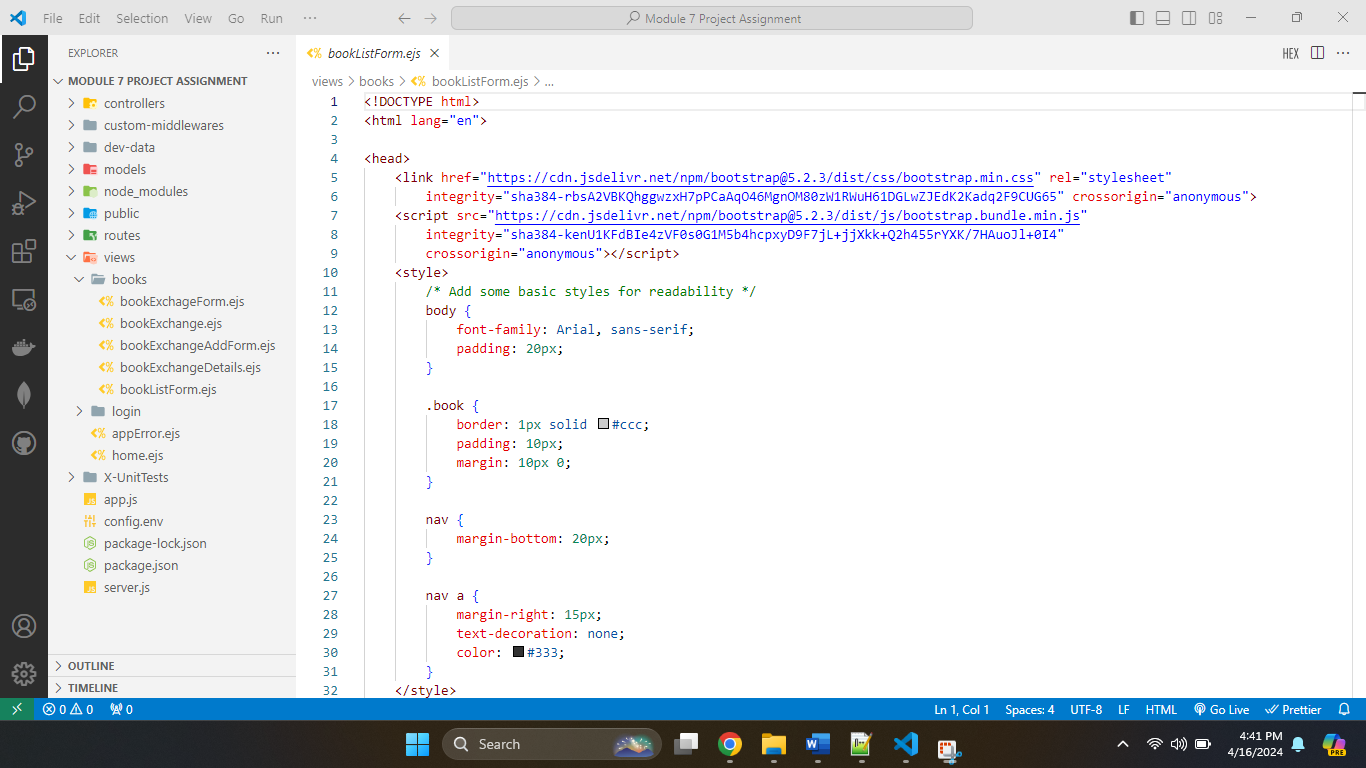
****

BookExchange.ejs is a webpage that displays available books from a book exchange platform. It begins with the definition of the document type and specifies the structure of the HTML document. The <head> element contains the page title and links to the Bootstrap CSS and JavaScript frameworks for style and functionality. Inline CSS styles also determine the look of components including the body, book entries, and navigation. The <body> portion includes JavaScript code that displays the current date and time, as well as the user's IP address obtained from an external API.

The header part provides the platform's title and navigation links. If a user is already signed in, it greets them and provides a logout link; otherwise, it offers registration and login links. The primary section lists the available books for exchange. It produces HTML components dynamically depending on book data supplied from server-side code (often via a templating engine like as EJS or Pug). It displays each book's title, author, genre, and description. If a person is logged in but does not own the book, a "Request Exchange" button appears. If no books are available, a notification noting this is presented. Finally, the footer contains a copyright notice for the platform.

Overall, this HTML code provides a user-friendly interface for browsing and engaging with available books on the book exchange network.

**bookListForm.ejs**

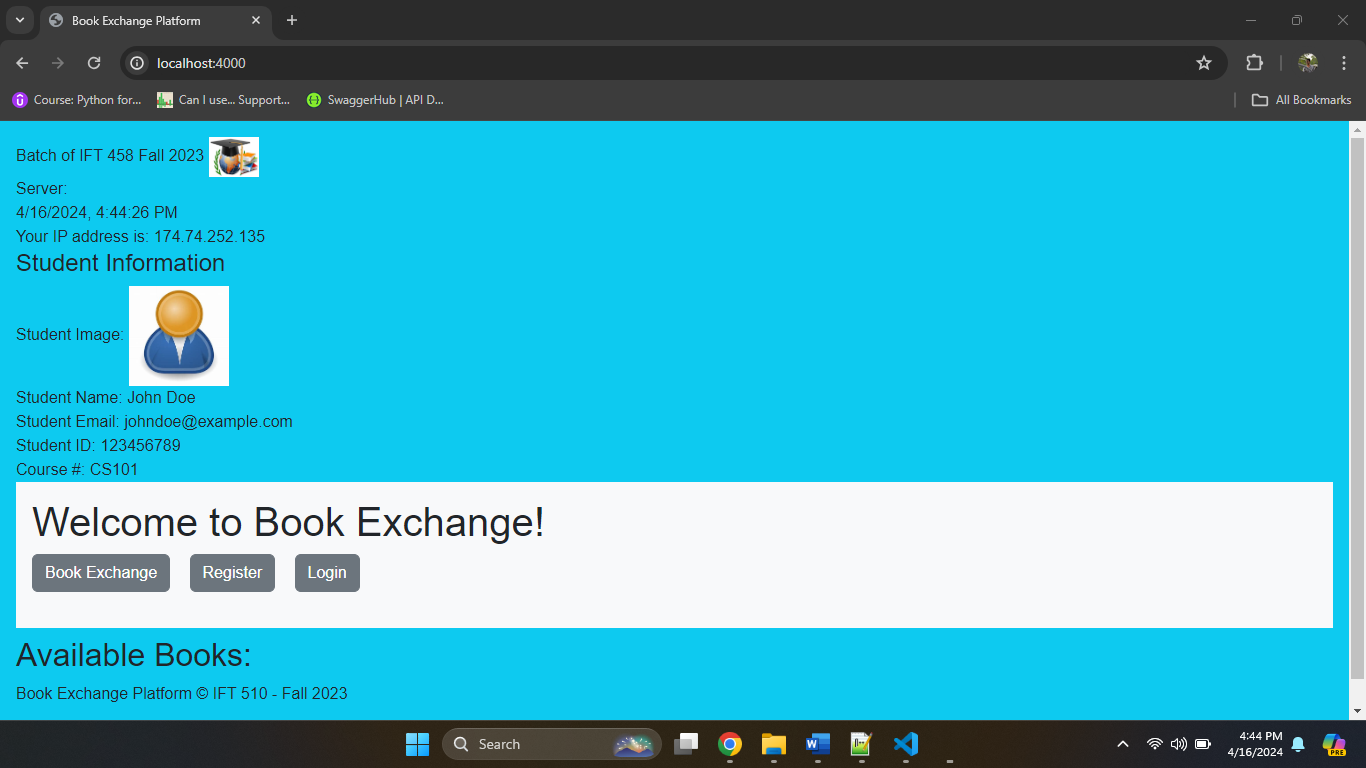


bookListForm.ejs is a website template that displays a list of books in a book exchange application. The <head> element imports Bootstrap CSS and JavaScript modules to improve the page's design and functionality. In addition, several unique CSS styles are created to enhance readability and layout. The <body> section contains JavaScript code that obtains and displays the current date and time, as well as the user's IP address, obtained via an API.

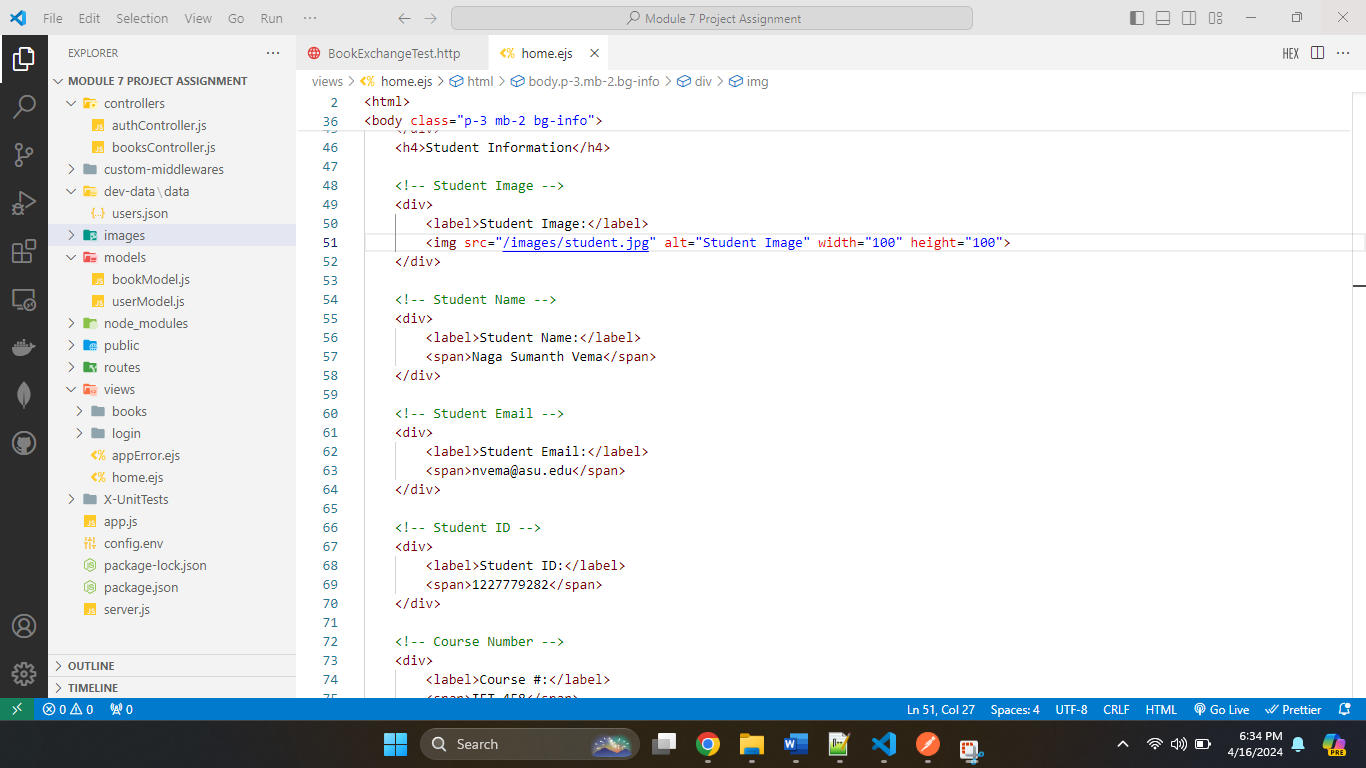
The <body> elements display the page's major content, such as the book list and a navigation menu for adding additional books. The book list is organized as an HTML table, with columns for title, author, description, exchange type, and status.

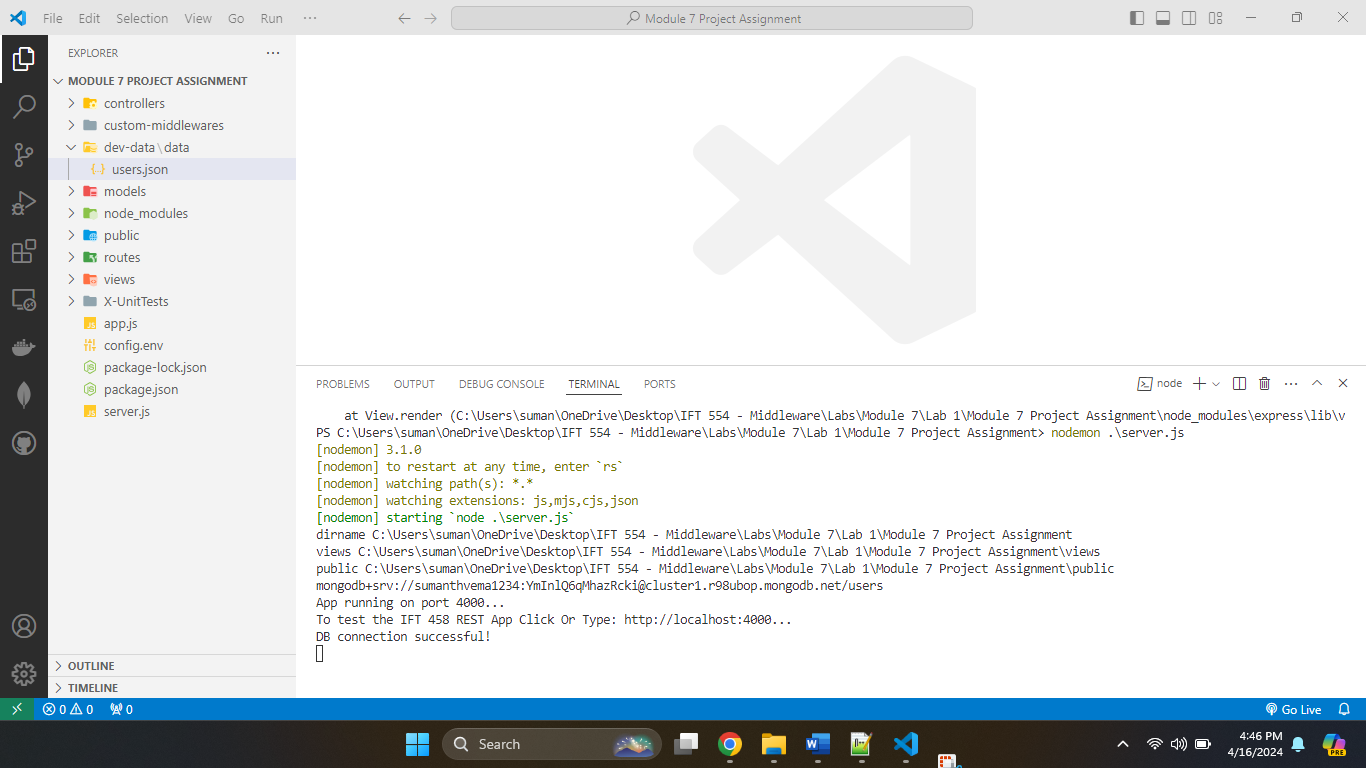
Book data is dynamically loaded into the HTML template via server-side templating (most likely EJS or something similar). The details of each book in the list are presented within a table row (<tr>), with links for updating and removing the book. Finally, a link to the home page is provided at the bottom of the page. Overall, this HTML template provides an organized and visually appealing interface for organizing and engaging with a number of  books in the book exchange application.

**Screenshot of the home page as displayed in the browser.**

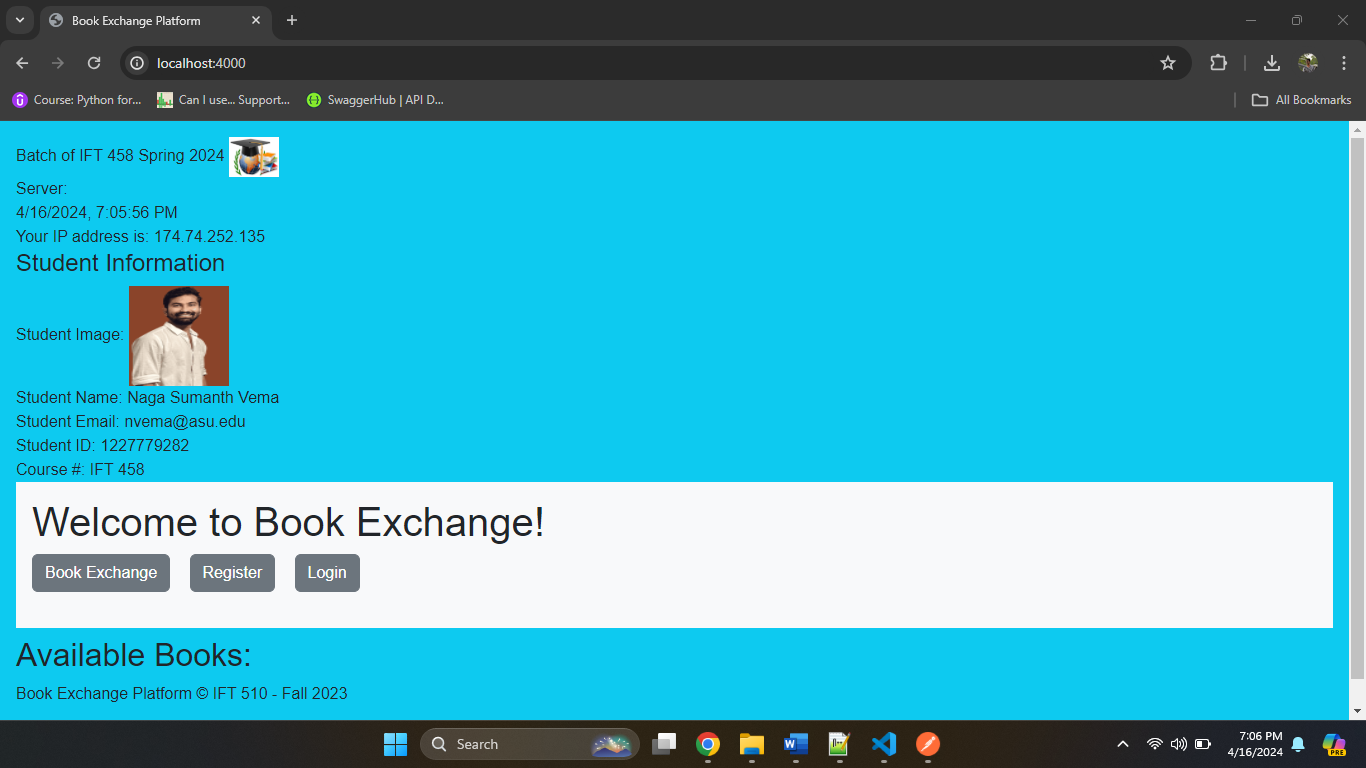
****

**Screenshots of all personalized user interface pages.**

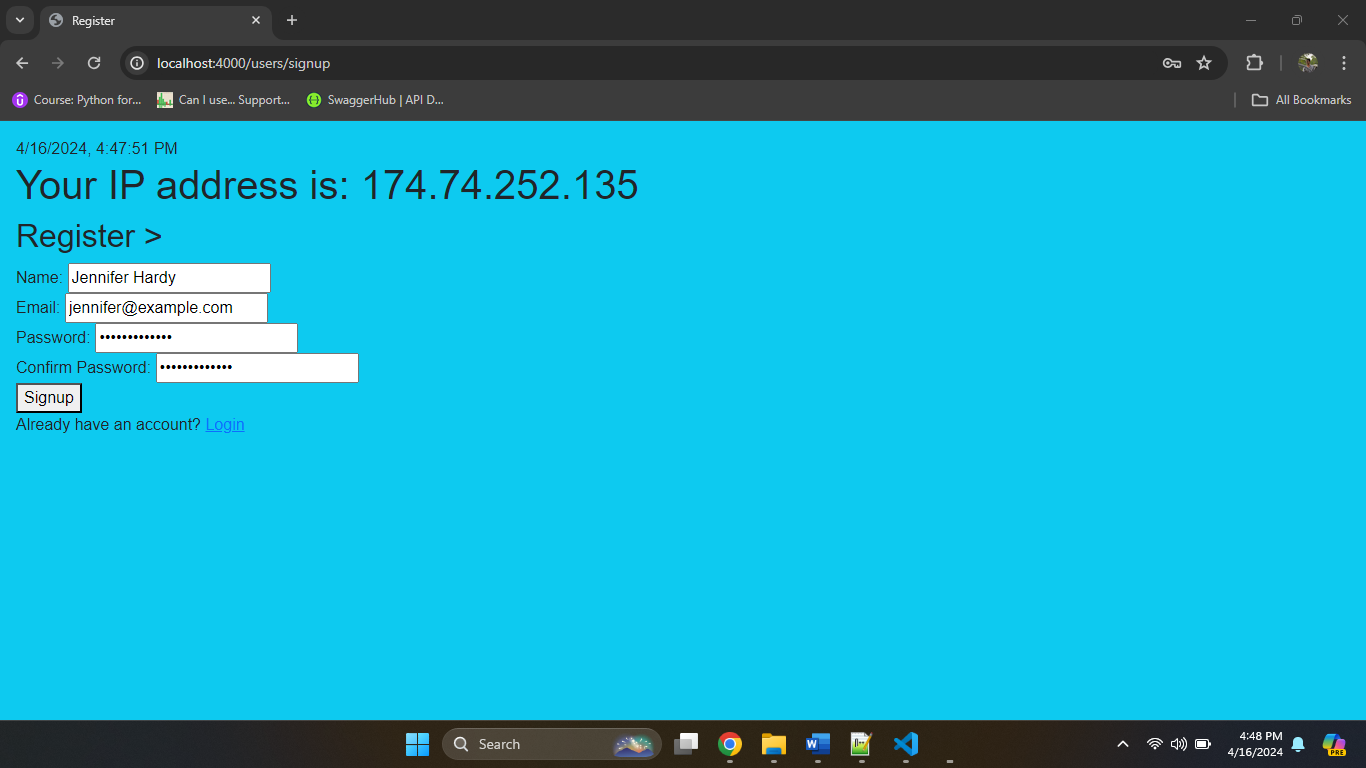
****

****

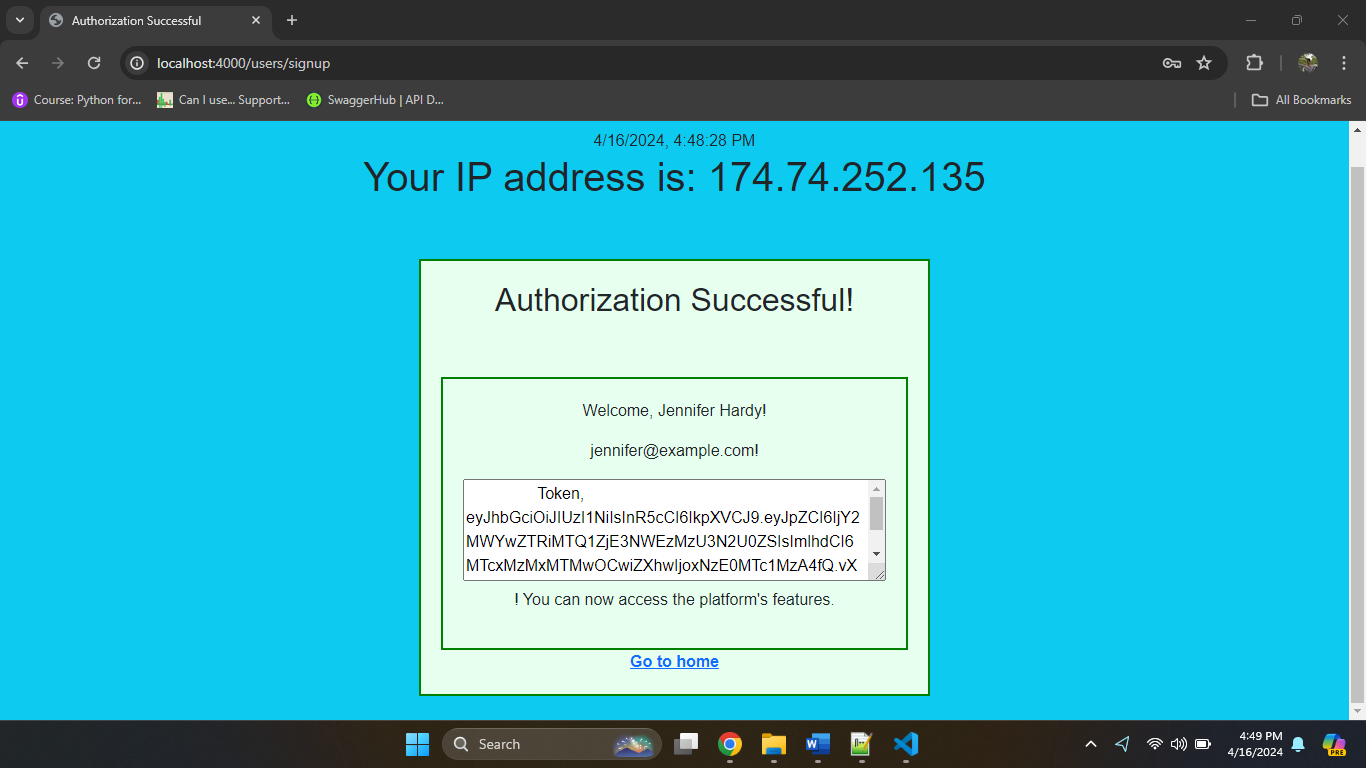
**Home Page**

****

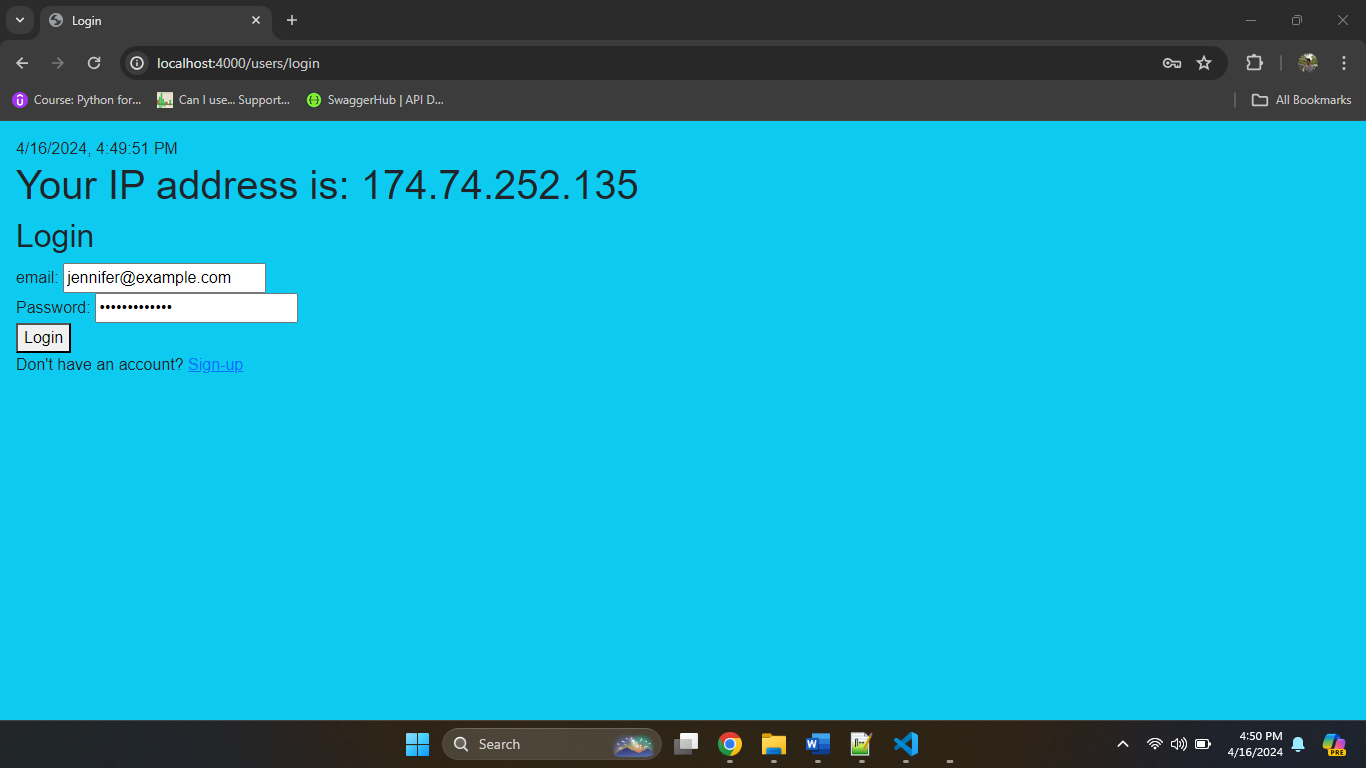
**Register Page**

****

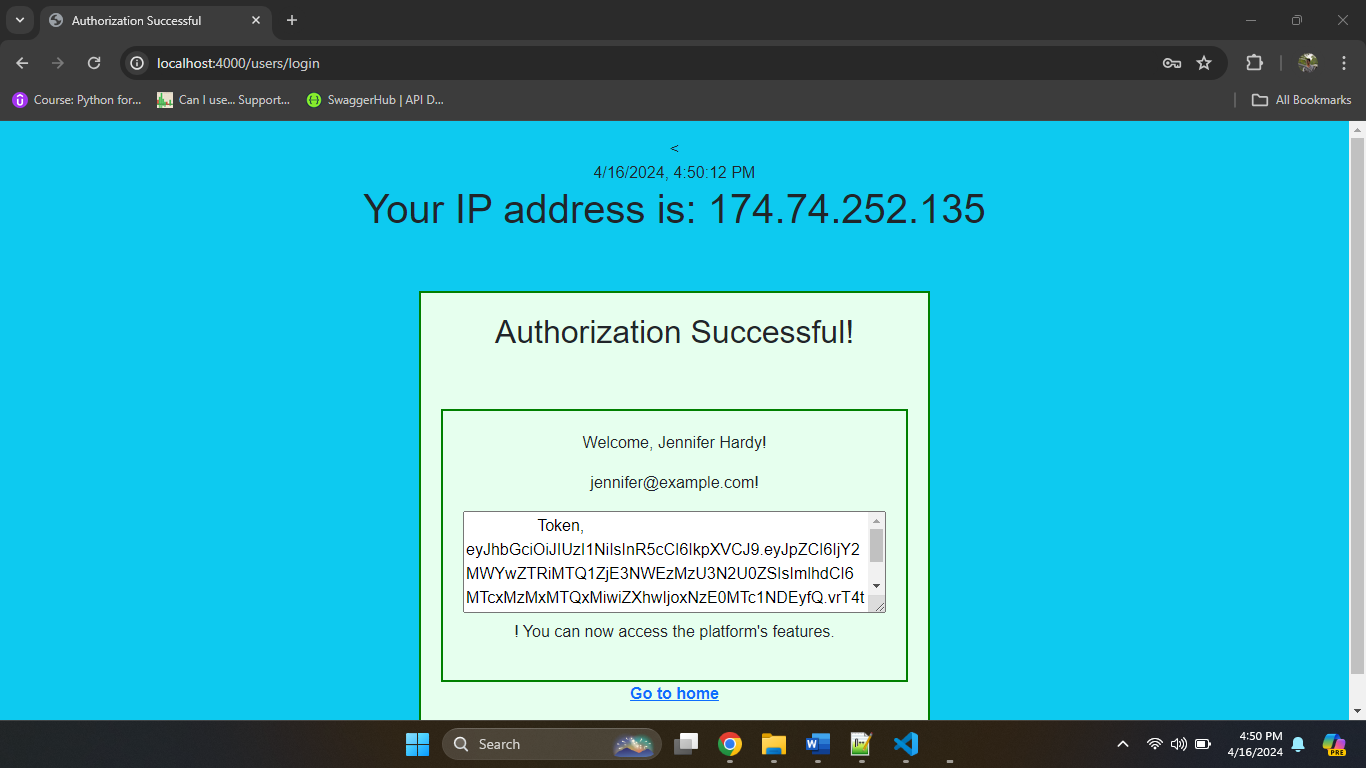
**After Successful registration**

****

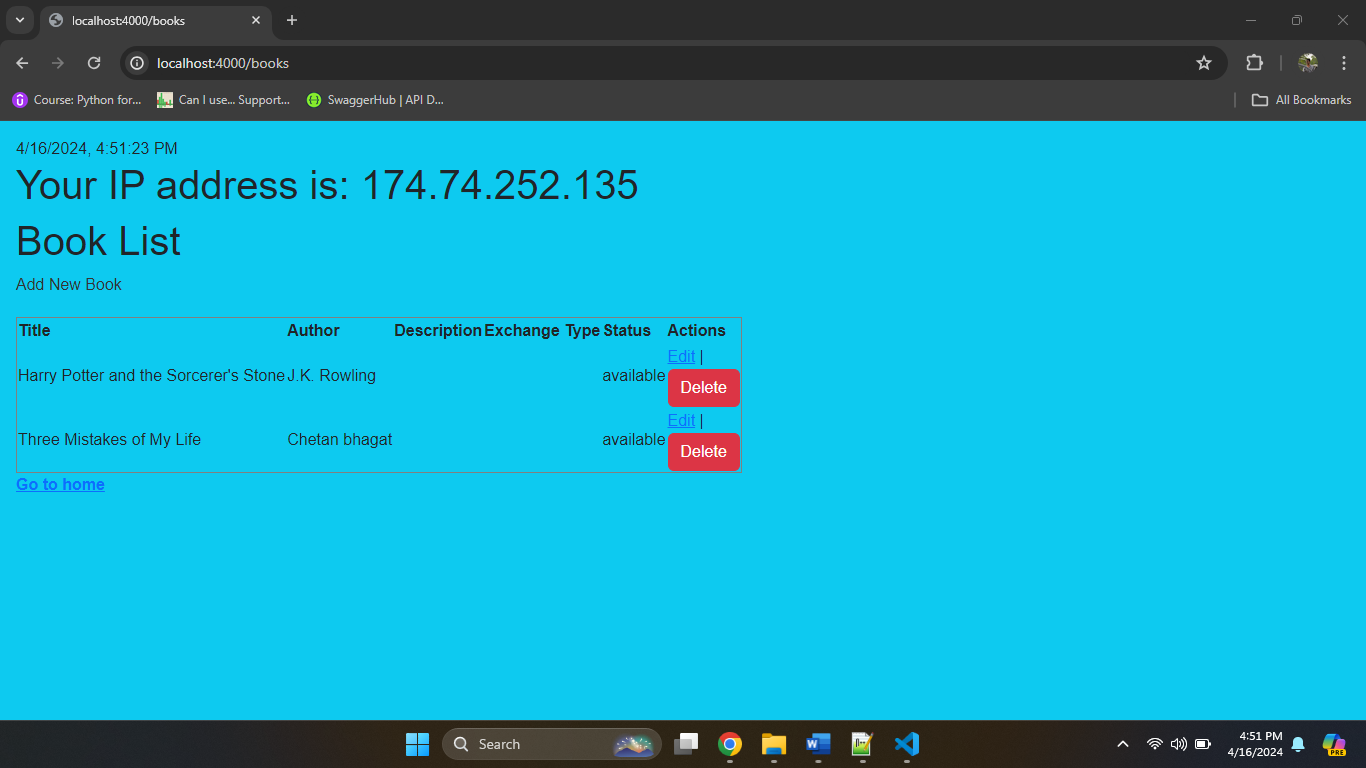
**Login page**

****

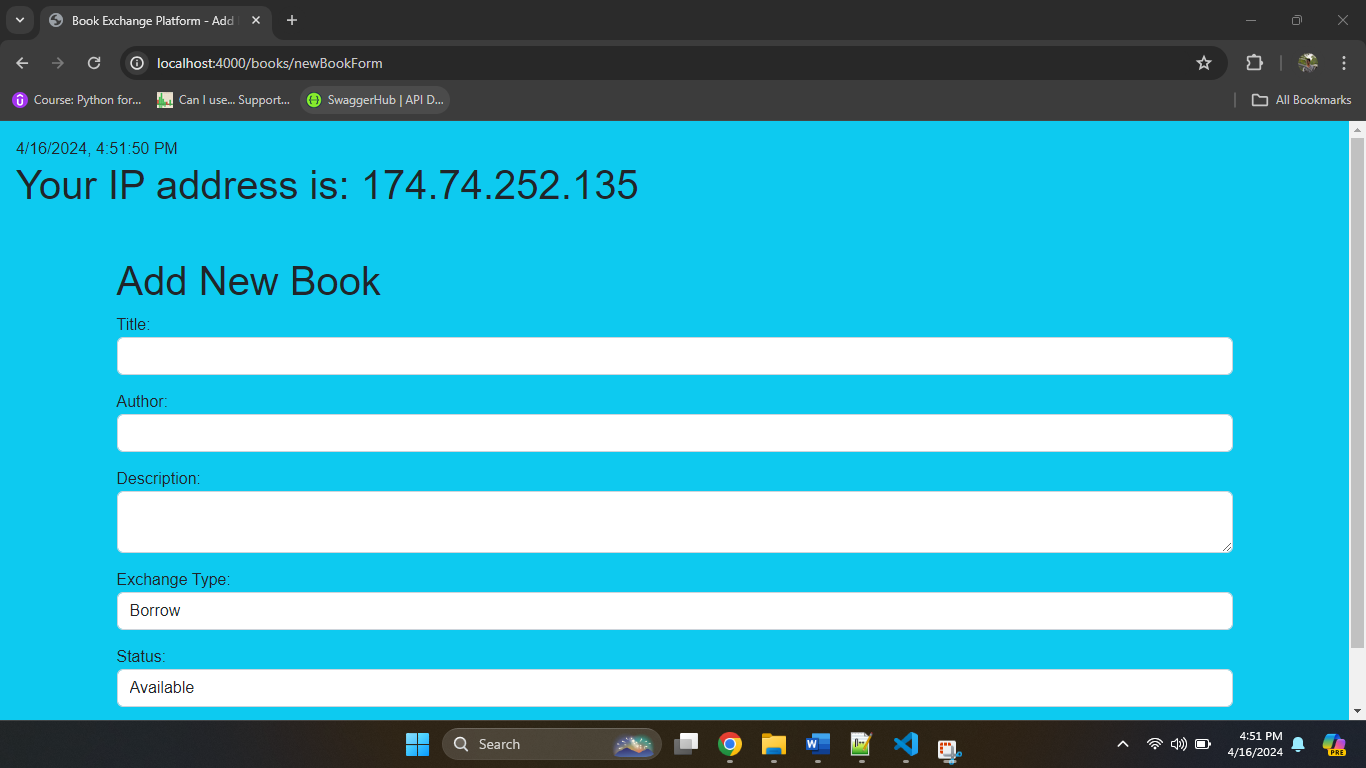
**After Successful login**

****

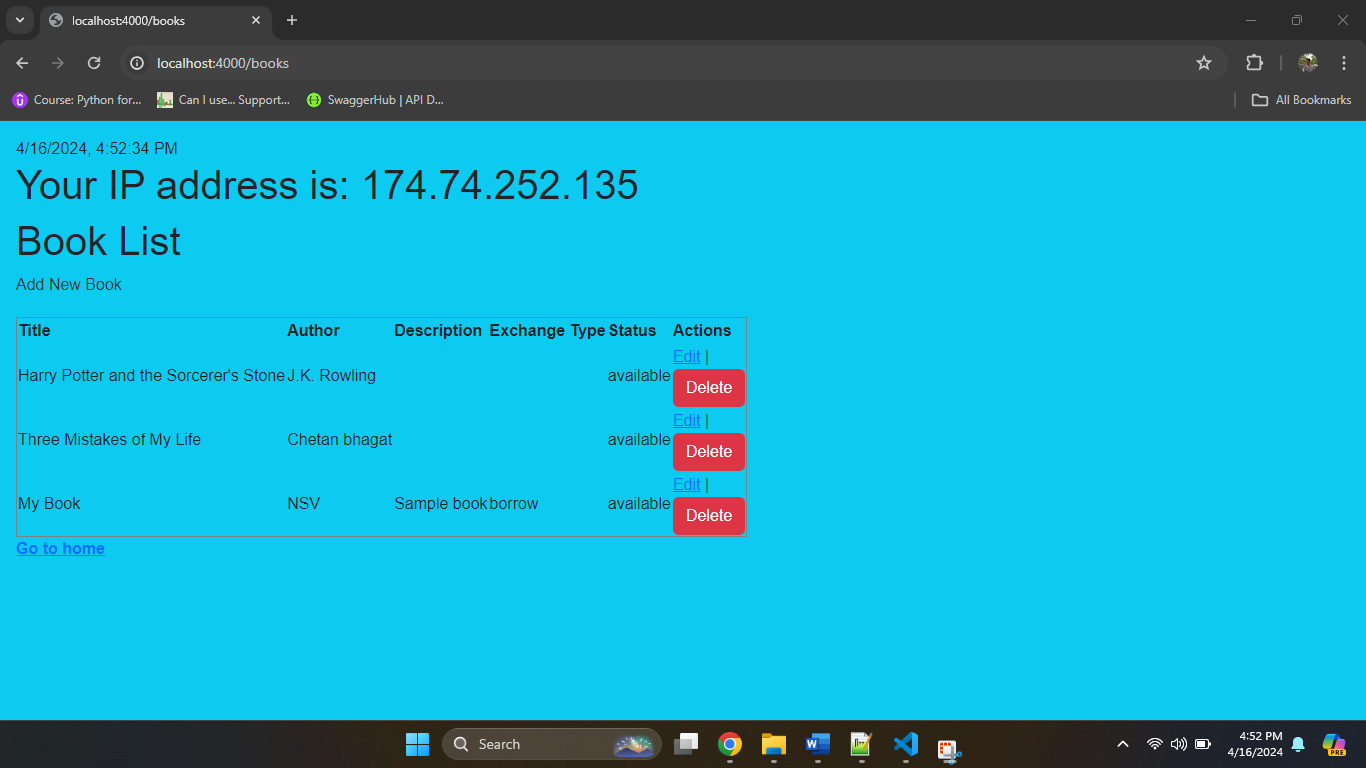
**Book Exchange page**

****

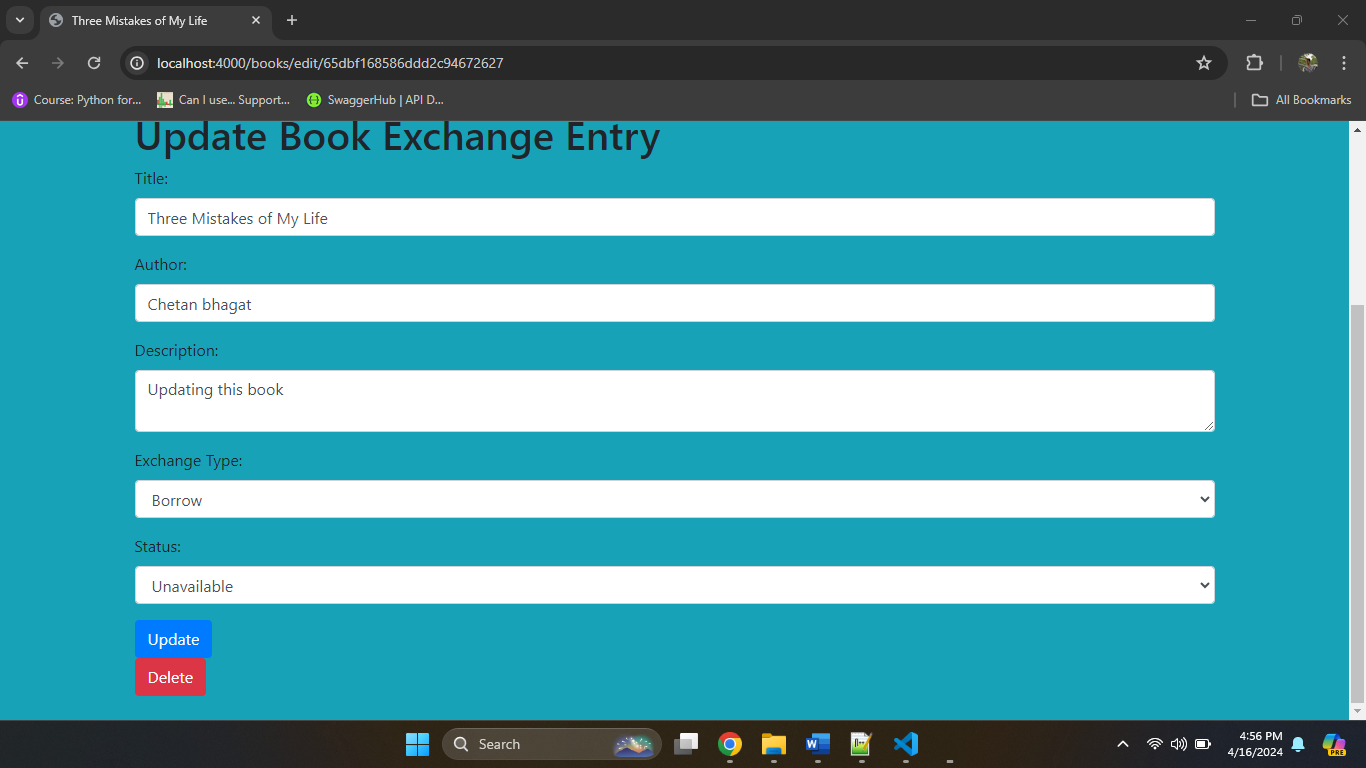
**Add new book page**

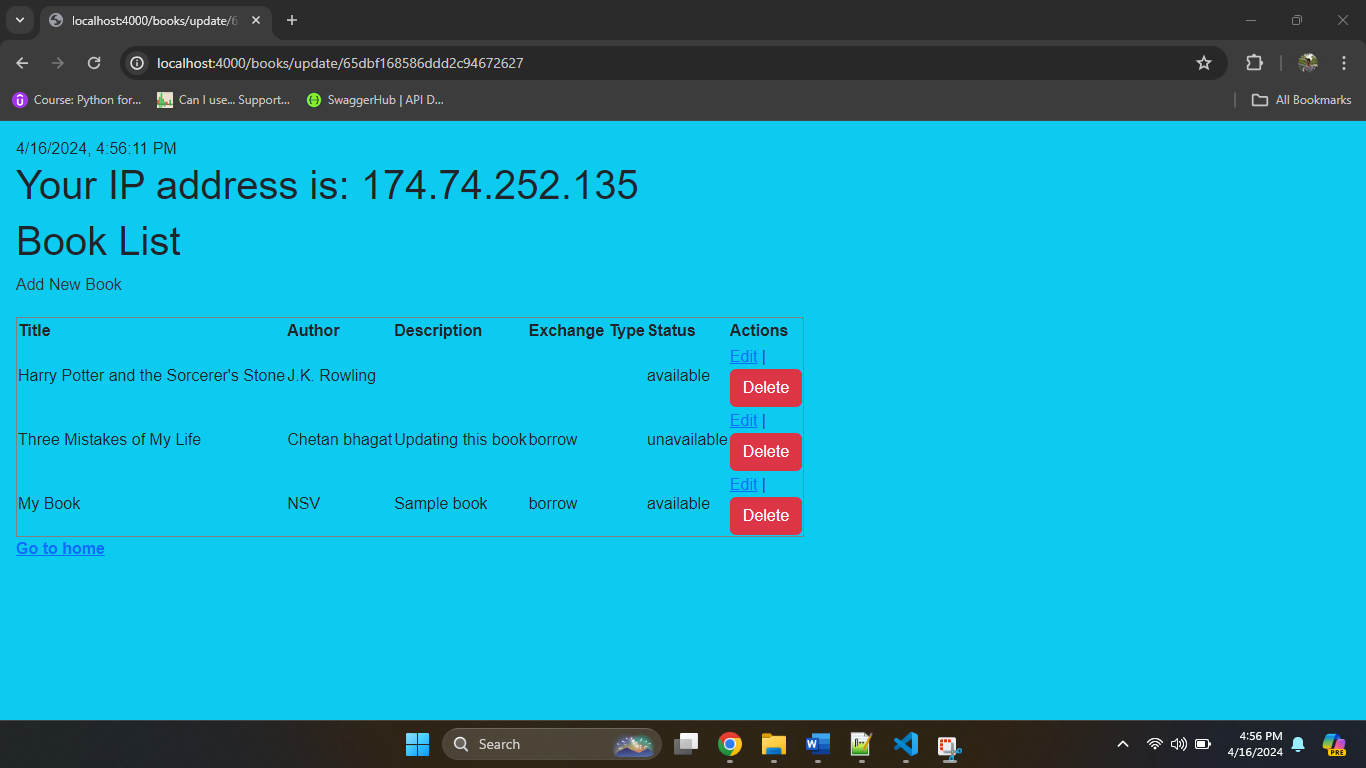
****

**After adding book**

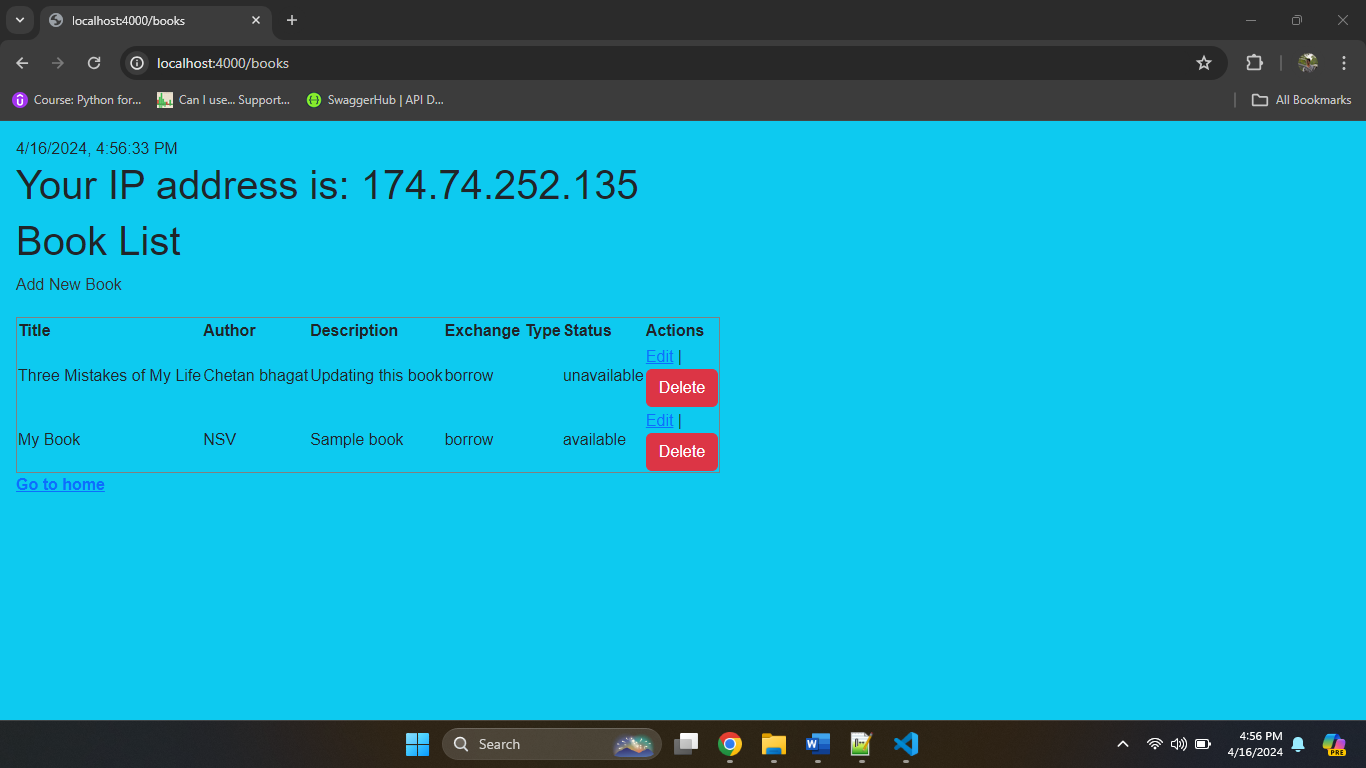
****

**Making updates to book exchange**

****

****

**After Deleting Harry potter book**

****

**Screenshots 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 | <https://localhost:4000/books/edit/65dbf168586ddd2c94672627> |
|  |  |  |

Register / Sign up