Ira A. Fulton Schools of Engineering
Arizona State University

Final Project

Diya Liza Varghese

IFT458&544: Middleware Prog & Database Sec (2023 Fall)

Prof. Dinesh Sthapit

December 01, 2023

The Polytechnic School

ABSTRACT

We are implementing a book exchange application (Devfolio, n.d.) on NodeJS, with the added functionality to sign up, login, add books to exchange, view the books available to exchange and perform different operations on the same. The application is intended to help book lovers to get a platform where they can list their own books, view what other books are available and get other books in exchange to their own books. The application prioritizes user friendly interactions, seamless and engaging platform for bibliophiles.

INTRODUCTION

Overview of the project

The project is about building an application and making the best usage of middlewares. We are creating an application on NodeJS. The UI is rendered using EJS. We are using middlewares, especially for authentication. The application is a perfect demonstration of MERN stack. MongoDB is used as the database (MongoDB, n.d.). The books are created and pushed into the MongoDB. The project utilizes Model View Controller architecture (MVC).

Problem Statement

In the contemporary landscape of reading, the lack of a centralized and user-friendly book exchange platform hinders readers from seamlessly connecting for meaningful exchanges. Current solutions fall short in providing personalized matching, secure transactions, and a vibrant community, highlighting the pressing need for a modern book exchange application that bridges these gaps and enhances the overall experience for avid readers.

Scope and Limitations

For this project, we try to include an all comprehensive book exchange platform usable. We prioritize user account creation and authentication, book item addition, updating, deletion etc. There is a proper structure created for the books and users as schema in the database. Users can view their listing and make changes to it, while select other's listed books for exchange as well. The application will be containerized and deployed using Docker.

At the same time, there is the limitation of the application would be that it does not do auto-matching or provide an opportunity to search the books that are required in exchange. Since the application runs on database support, it needs internet connectivity at all times. Additionally, the project may face limitations in addressing individual user preferences comprehensively and may not account for regional variations in reading preferences. It's essential to recognize these constraints to manage user expectations and guide the project's development within defined boundaries.

Justification of the technology stack used

1. Node.js:

Scalability and Efficiency: Node.js's event-driven architecture ensures scalability, allowing the application to handle concurrent requests efficiently (Sufiyan, 2023), providing a responsive experience for users engaged in book exchanges.

Single Language Usage: Node.js enables the use of JavaScript on both the client and server, streamlining development, and fostering a cohesive and consistent codebase.

2. MongoDB:

Flexible Schema: MongoDB's NoSQL, document-oriented structure accommodates the dynamic nature of book data, offering flexibility in schema design and facilitating easy updates to the database as the application evolves.

Scalability: MongoDB's horizontal scalability suits the potential growth of the application, ensuring optimal performance as the user base and data volume increase.

3. MVC Architecture:

Modular Development: The MVC architecture (GeeksforGeeks, 2022) provides a modular and organized development structure, enhancing code maintainability, readability, and the ability to scale the application's features systematically.

Separation of Concerns: MVC's clear separation of concerns between models, views, and controllers promotes code reusability and facilitates collaborative development among multiple team members.

4. EJS (Embedded JavaScript):

Dynamic Templating: EJS allows dynamic rendering of HTML templates, enabling the application to dynamically generate content based on user interactions, enhancing the overall user experience.

Seamless Integration: As a templating engine for Node.js, EJS seamlessly integrates with the application, promoting efficient server-side rendering and reducing client-side processing.

5. JWT Tokens:

Secure Authentication: JSON Web Tokens (JWT) offer a secure and efficient method for user authentication, ensuring that user data is transmitted securely between the client and server during authorization processes.

Stateless Sessions: JWT's stateless nature minimizes server-side storage requirements and enhances scalability by eliminating the need for session information storage.

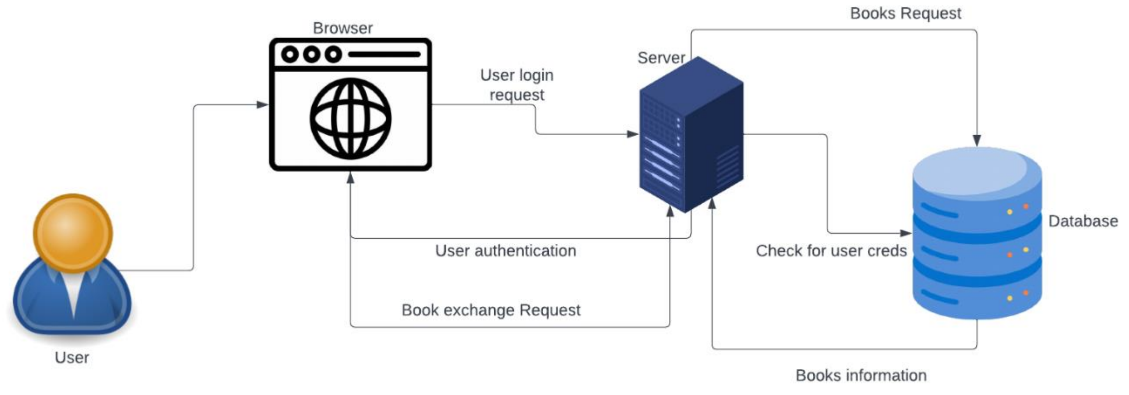
6. Docker:

Containerization for Consistency and Portability: Docker (Build and Run a Node.js App in a Container, 2021) enables the encapsulation of an application and its dependencies into a single, lightweight container. This containerization ensures consistency across different environments, making it easier to develop, test, and deploy the book exchange application.

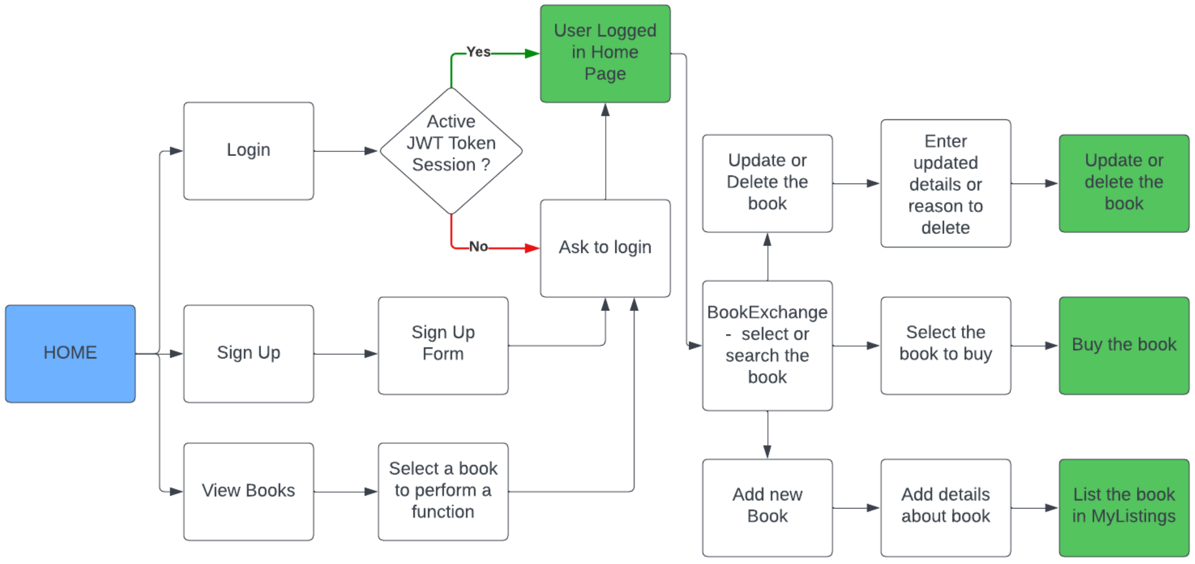
Efficient Deployment and Scalability: Docker facilitates seamless deployment and scaling of applications. With Docker containers, you can easily deploy the book exchange application across different environments without worrying about compatibility issues or complex setup procedures.

PROJECT DESIGN

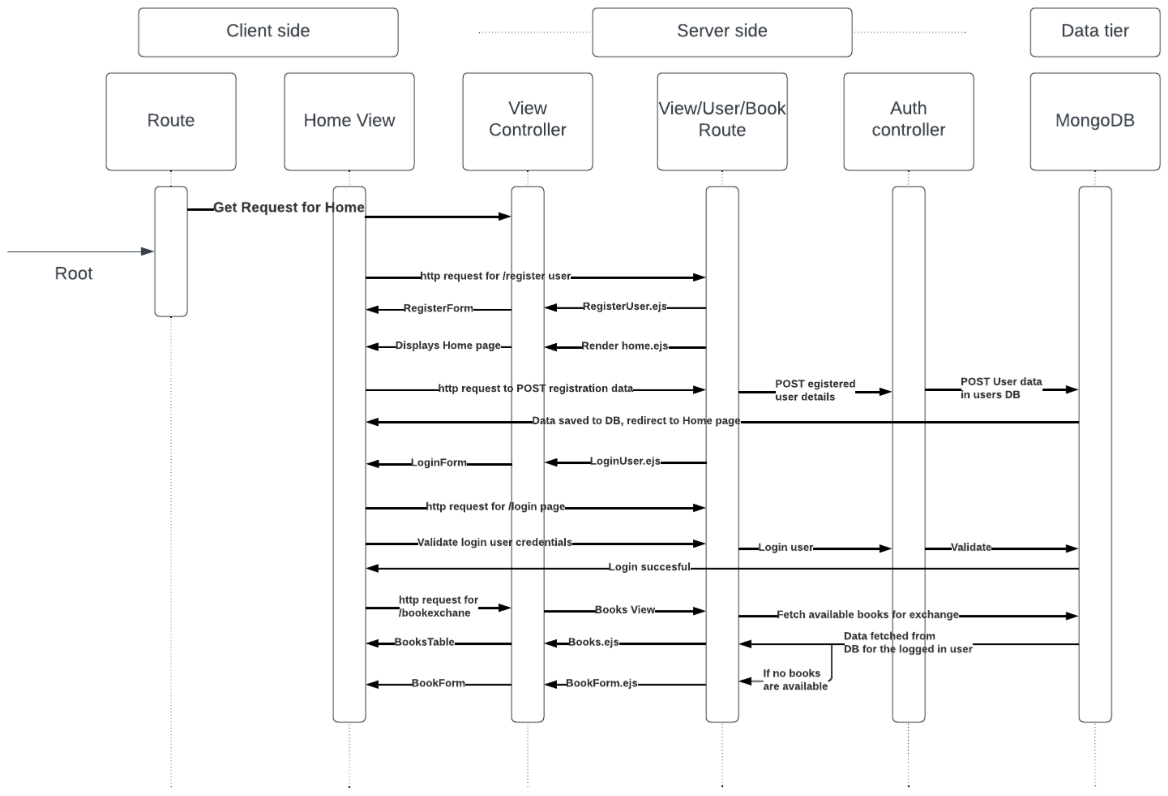
1. Architecture Diagrams



2. Flowchart


HOME
Login
Sign Up
View Books
Active JWT Token Session ?
Yes
No
User Logged in Home Page
Ask to login
Sign Up Form
Select a book to perform a function
Update or Delete the book
Enter updated details or reason to delete
Update or delete the book
BookExchange - select or search the book
Select the book to buy
Buy the book
Add new Book
Add details about book
List the book in MyListings

3. Sequence Diagram


Client side
Server side
Data tier
Route
Home View
View Controller
View/User/Book Route
Auth controller
MongoDB
Root
Get Request for Home
http request for /register user
RegisterForm
RegisterUser.ejs
Displays Home page
Render home.ejs
http request to POST registration data
POST egistered user details
POST User data in users DB
Data saved to DB, redirect to Home page
LoginForm
LoginUser.ejs
http request for /login page
Validate login user credentials
Login user
Validate
Login succesful
http request for /bookexchane
Books View
Fetch available books for exchange
BooksTable
Books.ejs
Data fetched from DB for the logged in user
BookForm
BookForm.ejs
If no books are available

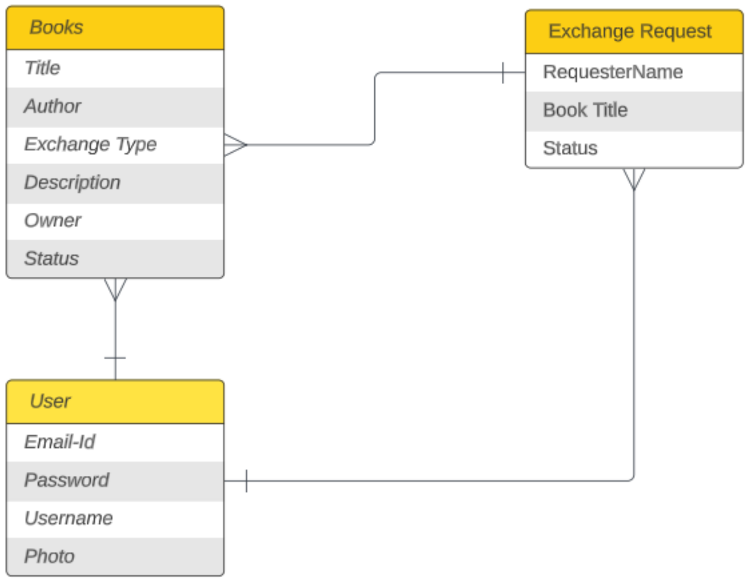
Authentication Mechanism:

- User Registration: Users create accounts by providing a unique username, email address, and password. Passwords are securely hashed and stored using a strong and adaptive hashing algorithm (e.g., bcrypt).
- User Login: Users provide credentials (username/email and password) to the authentication server. Upon successful authentication, the server generates a JWT containing user information and sends it to the client.
- JWT Structure: The JWT payload contains user information (e.g., user ID, roles). Tokens are signed using a secret key, ensuring their integrity.

- Token Expiration: Tokens have an expiration time, enhancing security. Refresh tokens can be used to obtain a new access token without requiring the user to re-enter credentials.
- Multi-Factor Authentication (MFA): Optional but highly recommended for enhanced security. Users can enable MFA, and MFA-related data can be included in the JWT payload.

Authorization Mechanism:

- Token Verification: The server verifies incoming requests by checking the validity and signature of the JWT. User roles and permissions are extracted from the token.
- Role-Based Access Control (RBAC): Define roles within the JWT payload (e.g., user, admin) and assign specific permissions. Access to resources (e.g., API endpoints) is authorized based on the user's role.
- Ownership-Based Authorization: Users can only modify or delete resources they own (e.g., books). Exchange requests and transactions are authorized based on ownership included in the token.
- Fine-Grained Authorization Policies: Implement specific authorization policies within the application based on the user's roles and other attributes present in the JWT.



Entity-Relationship Diagram

Entities:

Books:

Title (String): The title of the book.

Author (String): The author of the book.

Exchange Type (String): Specifies the type of exchange (e.g., sell, exchange).

Description (String): A description of the book.

Owner (Foreign Key to User): The user who owns the book.

Status (String): Represents the status of the book (e.g., available, reserved).

Exchange Request:

RequesterName (String): The name of the user making the exchange request.

Book Title (Foreign Key to Books): The title of the requested book.

Status (String): Indicates the status of the exchange request (e.g., pending, accepted, declined).

User:

Email-Id (String): The email address of the user.

Password (String): The user's password.

Username (String): The username chosen by the user.

Photo (String): URL or reference to the user's profile photo.

Relationships:

Books - Exchange Request:

One-to-Many Relationship: A book can have multiple exchange requests, but each request is associated with only one book. This is represented by a line connecting the Books and Exchange Request entities, with a crow's foot indicating the "many" side.

Books - User:

Many-to-One Relationship: Many books can be owned by one user. This is represented by a line connecting the Books and User entities, with a line pointing to the User entity.

Exchange Request - User:

Many-to-One Relationship: Many exchange requests can be made by one user. This is represented by a line connecting the Exchange Request and User entities, with a line pointing to the User entity.

4. Pseudocode / Code Snippets

Book Schema definition:

```
const mongoose = require('mongoose');

const bookExchangeSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    author: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    exchangeType: {
        type: String,
        enum: ['borrow', 'trade'],
        required: true
    },
    owner: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    },
    status: {
        type: String,
        enum: ['available', 'unavailable'],
        default: 'available'
    }
}, { timestamps: true });

const BookExchange = mongoose.model('BookExchange', bookExchangeSchema);

module.exports = BookExchange;
```

User Model Definition

```
const mongoose = require('mongoose');
// import bcryptjs
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please tell us your name!']
  },
  email: {
    type: String,
    required: [true, 'Please provide your email'],
    unique: true,
    lowercase: true
  },
  photo: String,
  role: {
    type: String,
    enum: ['client', 'staff', 'student', 'admin'],
    default: 'client'
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: 8,
    select: false
  },
  // validate password
  passwordConfirmation: {
    type: String,
    required: [true, 'Please confirm your password'],
    // validate will only work on save or creating the user
    validate:{
        validator: function(pwd){
          return pwd === this.password;
        },
        message: " the password confirmation did not match"
    }
  },
  passwordChangedAt: Date,
  passwordResetExpires: Date,
  active: {
    type: Boolean,
    default: true,
    select: false
  },
  roles:{
    type: String,
    default:'client'// authorization
  }
});

// add encrypting the password middleware
userSchema.pre('save', async function(next){
  if(!this.isModified('password')){
    return next;
  }
  // if new and modified
  // npm install bcryptjs
  this.password = await bcrypt.hash(this.password, 12); // how intensive the CPU
will be
  this.passwordConfirmation = undefined; // we dop not want to store in database
  next();
});
//model instance method will be available to all the instance of the user.
userSchema.methods.isPasswordMatch = async (userSuppliedPassword,
currentHashedPasswordInDB)=>{
  return await bcrypt.compare(userSuppliedPassword, currentHashedPasswordInDB)
}
const User = mongoose.model('User', userSchema);
```

```
module.exports = User;
```

BooksController logic –

POST:

```
// CREATE a new book
exports.createBook = async (req, res) => {
    try {
        const book = new Book({
        title: req.body.title,
            author: req.body.author,
            description: req.body.description,
            exchangeType: req.body.exchangeType,
            owner: req.user._id,  // Assuming req.user contains the authenticated
user
            status: req.body.status // Or default to 'available' as per your
schema
        });
        const newBook = await book.save();
        res.redirect('/books'); // assuming '/books' is where you list all books
    } catch (err) {
        res.status(400).render('appError', { title: 'Create Book', user:
req.user, errors: err.errors });
    }
};
```

GET:

```
exports.getBooks = async (req, res) => {
    try {
        const books = await Book.find();
        res.render('./books/bookListForm', { title: 'All Books', user: req.user,
books });
    } catch (err) {
        res.status(500).render('appError', { message: err.message });
    }
};

// GET a single book by ID
exports.getBookById = async (req, res) => {
    try {
            const book = await Book.findById(req.params.id);
            if (!book) {
                return res.status(404).render('error', { message: 'Book not
found' });
            }
            res.render('./books/bookExchangeDetails', { book });
    } catch (err) {
        res.status(500).render('error', { message: err.message });
    }
};
```

PUT:

```
exports.updateBookById = async (req, res) => {
    try {
      const book = await Book.findByIdAndUpdate(req.params.id, req.body, {
        new: true,
      });
      if (!book) {
        return res.status(404).render("error", { message: "Book not found" });
      }
      const updatedBooks = await Book.find();
      res.render(`./books/bookListForm`, {books: updatedBooks}); // Redirect to
book detail page
      //res.redirect(`/books/${req.params.id}`); // Redirect to book detail page
    } catch (err) {
        res.status(500).render('error', { message: err.message });
    }
};

exports.updateBookAndReturn = async (req, res) => {
    try {
      const book = await Book.findByIdAndUpdate(req.params.id, req.body, {
        new: true,
      });
      if (!book) {
        return res.status(404).json({ message: "Book not found" });
      }

      // Send the updated book details in the response
      res.json(book);

    } catch (err) {
      // Handle errors appropriately
      res.status(500).json({ message: err.message });
    }
  };
```

DELETE:

```
exports.deleteBook = async (req, res) => {
    try {
            const book = await Book.findByIdAndRemove(req.params.id);
            if (!book) {
                return res.status(404).json({ message: "Book not found" });
            }
            res.redirect('/books'); // Redirect to all books list
    } catch (err) {
        res.status(500).render('error', { message: err.message });
    }
};

exports.deleteBookById = async (req, res) => {
    try {
            const book = await Book.findByIdAndRemove(req.params.id);
            if (!book) {
                return res.status(404).json({ message: "Book not found" });
            }
            return res.json({'message':'success'}); // Redirect to all books list
    } catch (err) {
        return res.status(500).json({ message: "Error" })
    }
};
```

User Controller Logic:

Sign Up:

```
exports.signup = async (req, res, next) => {
  const { name, email, password, passwordConfirmation } = req.body;
  try {
    const newUser = await User.create({
      name,
      email,
      password,
      passwordConfirmation
    });
    //and then send the token
    createSendToken(newUser, 201, res);// this is called Authorization
  } catch (error) {
    res.status(400).render('./login/registerForm', {
      errorCode: 400,
      errorMessage: error.message
    });
  }
};
```

Create Sign Token

```
const createSendToken = (user, statusCode, res) => {
  const token = signToken(user._id);
  const cookieOptions = {
    expires: new Date(
      Date.now() + process.env.JWT_COOKIE_EXPIRES_IN * 24 * 60 * 60 * 1000
    ),
    httpOnly: true
  };
  if (process.env.NODE_ENV === 'production') cookieOptions.secure = true;

  res.cookie('jwt', token, cookieOptions);

  // Remove password from output
  user.password = undefined; // very important
  res.render( './login/authorizationSuccess', { user: user, token: token});
};
```

Sign Token

```
const signToken = id => {
  // document reference https://www.npmjs.com/package/jsonwebtoken
  try {
    const jwtToken = jwt.sign({ id }, process.env.JWT_SECRET, {
      //noTimestamp:true,
      expiresIn: process.env.JWT_EXPIRES_IN
    });
    console.log;
    return jwtToken;
  } catch (error) {
     res.render('./users/login', { title: 'Login', user: undefined, token: 
undefined });
  }
};
```

Login:

```
exports.login = async (req, res, next) => {
  const { email, password } = req.body;
  // 1) Check if email and password exist
  if (!email || !password) {

    // the get verb does not send the body. so there is no body object
    res.status(401).render('./login/loginForm', {
      errorCode: 401,
      errorMessage: 'Please provide email and password!'
    });
    return

  }
  // 2) Check if user exists && password is correct
  const user = await User.findOne({ email }).select('+password');

  if (!user || !(await user.isPasswordMatch(password, user.password))) {
    res.status(401).render('appError', {
      errorCode: 401,
      errorMessage: 'Incorrect email and password!'
    });
  }

  // 3) If everything ok, send token to client
  try {
    createSendToken(user, 200, res);
  }catch(error){
    res.status(401).render('./login/loginForm', {
      errorCode: 401,
      errorMessage: 'Incorrect email and password!'
    });
  }

};
```

LogOut:

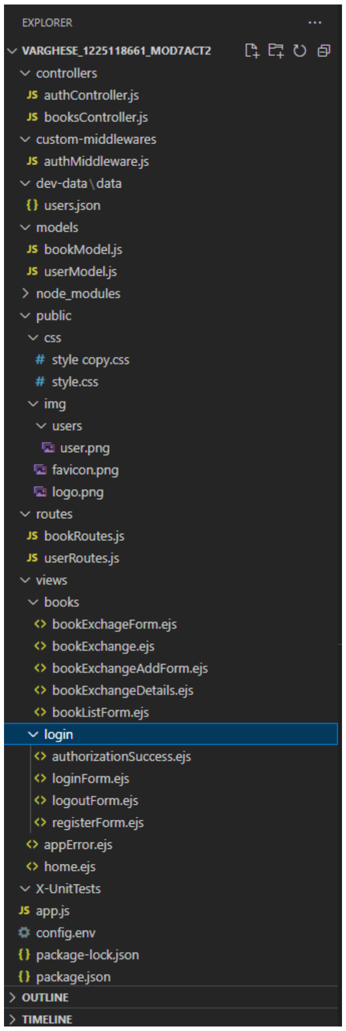
```
exports.logout = (req, res) => {
  res.clearCookie('jwt', { httpOnly: true, secure: process.env.NODE_ENV ===
'production' });
  res.render('./login/logoutSuccess', { title: 'Logout' });
};
```

5. API Specifications (if applicable).

Feature	Method	URL End Point (the following are just the sample) You must update the URLs with the correct endpoints)
Register / Sign up	POST	http://localhost:4000/users/signup

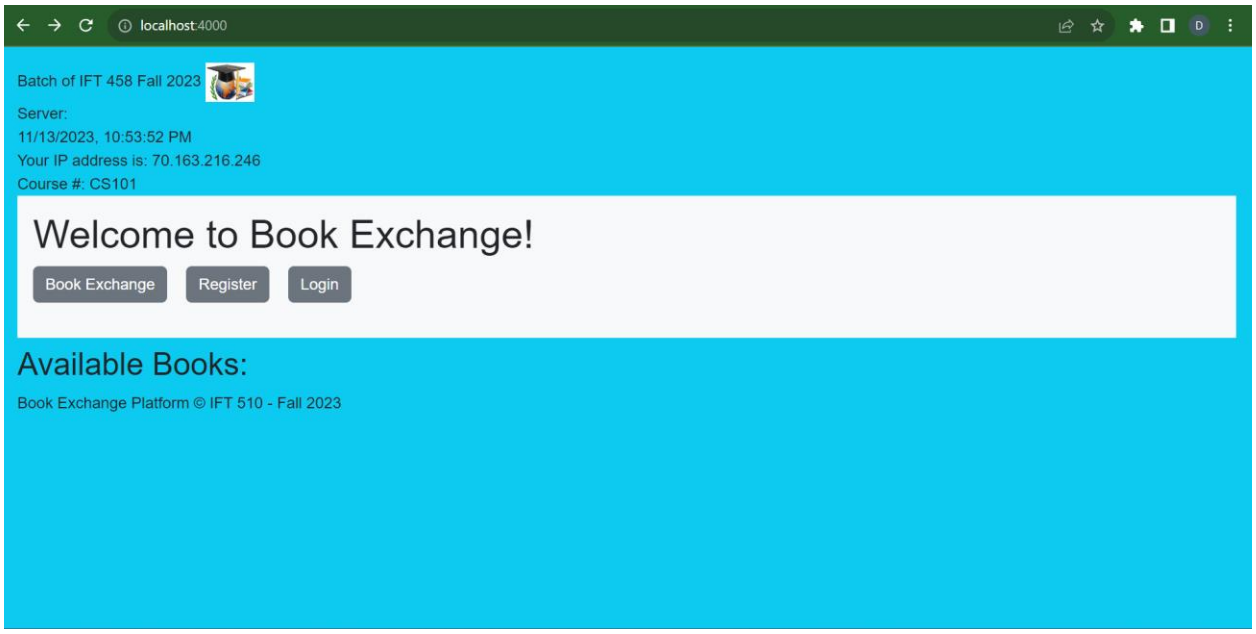
Login	POST	http://localhost:4000/users/login
View All Books	GET	http://localhost:4000/books
Add Book	POST	http://localhost:4000/books
Get a specific book	GET	http://localhost:4000/books:id
Get the Add book form	GET	http://localhost:4000/books/newBookForm
Get the Update form	GET	http://localhost:4000/books/edit/:id
Update the book	POST	http://localhost:4000/books/update/:id
Delete the book	POST	http://localhost:4000/books/delete/:id
Get the login form	GET	http://localhost:4000/users/login
Get the signup form	GET	http://localhost:4000/users/signup
Get the logout form	GET	http://localhost:4000/users/logoutUser

SCREENSHOTS



```
EXPLORER
VARGHESE_1225118661_MOD7ACT2
  controllers
    authController.js
    booksController.js
  custom-middlewares
    authMiddleware.js
  dev-data \ data
    users.json
  models
    bookModel.js
    userModel.js
  node_modules
  public
    css
      style copy.css
      style.css
    img
      users
        user.png
      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
  config.env
  package-lock.json
  package.json
OUTLINE
TIMELINE
```

Figure 1-Project Structure


localhost:4000

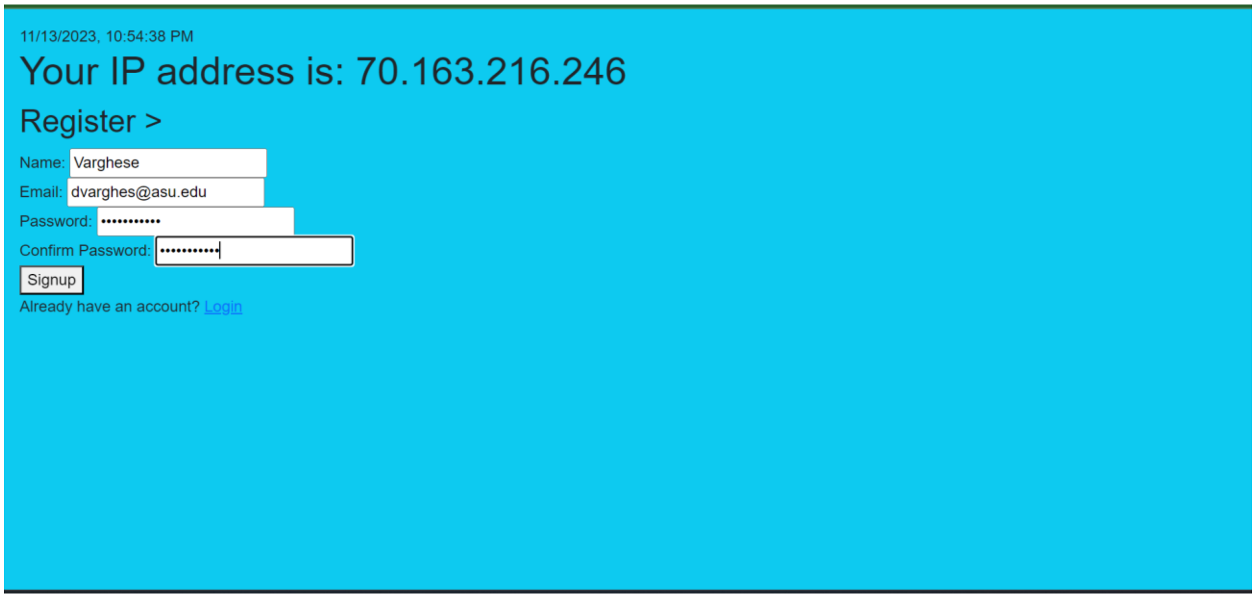
Batch of IFT 458 Fall 2023
Server:
11/13/2023, 10:53:52 PM
Your IP address is: 70.163.216.246
Course #: CS101

Welcome to Book Exchange!

Available Books:

Book Exchange Platform © IFT 510 - Fall 2023

Figure 2-Welcome Page


11/13/2023, 10:54:38 PM

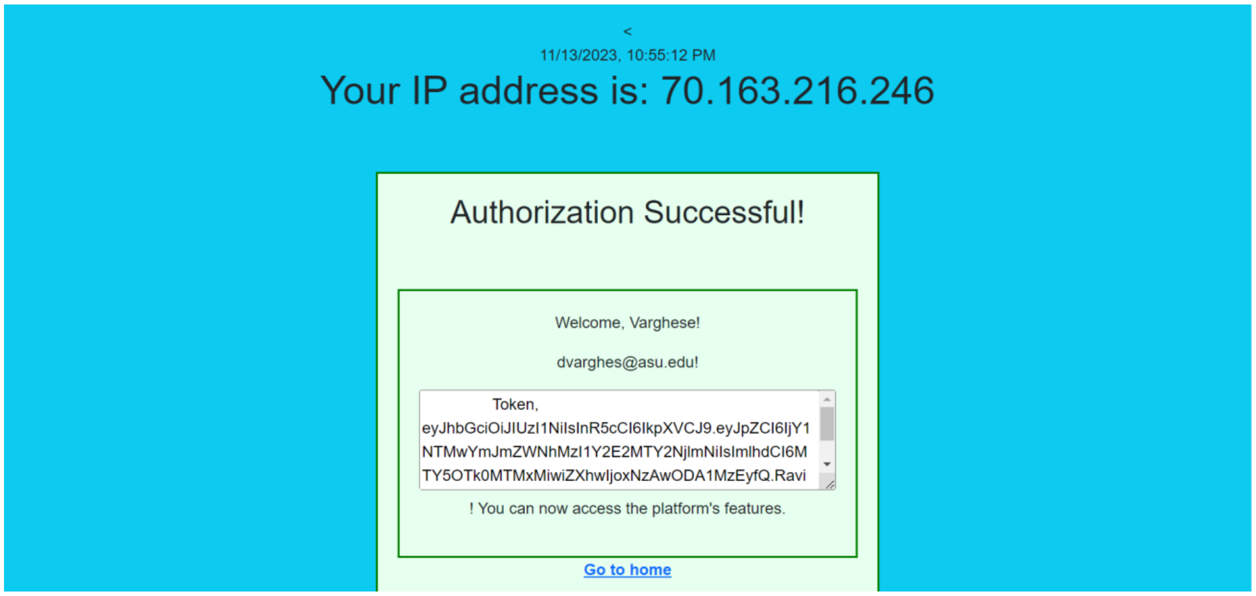
Your IP address is: 70.163.216.246

Register >

Name: Varghese
Email: dvarghes@asu.edu
Password: •••••••••••
Confirm Password: •••••••••••

Already have an account? Login

Figure 3-Register User


<
11/13/2023, 10:55:12 PM

Your IP address is: 70.163.216.246

Authorization Successful!

Welcome, Varghese!

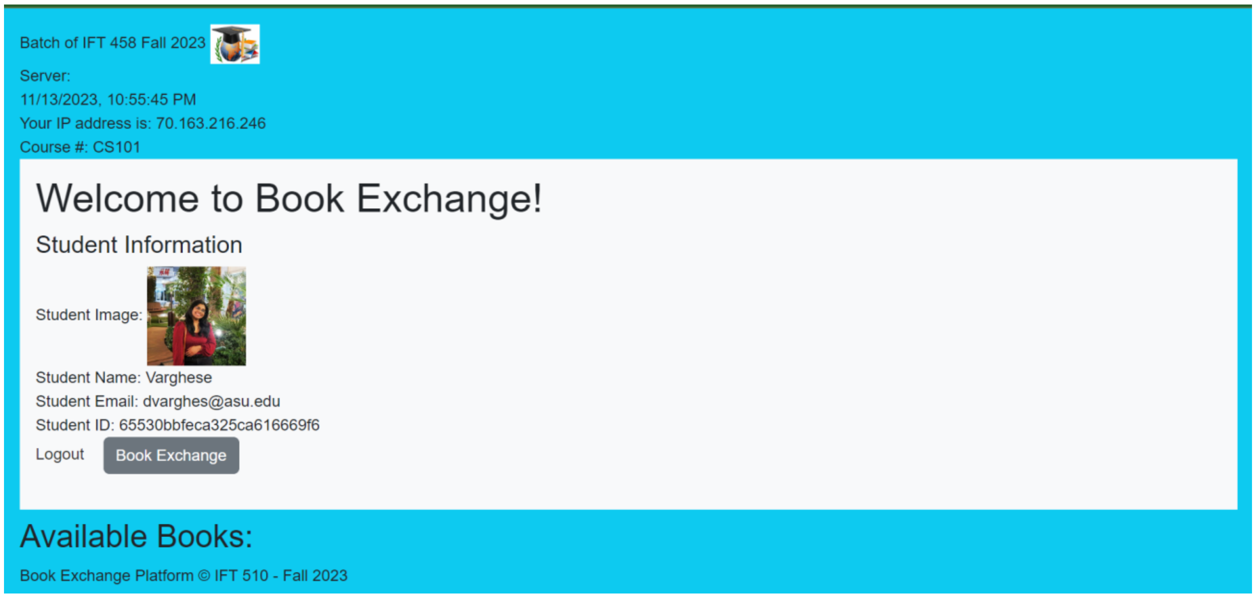
dvarghes@asu.edu!

Token,
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTMwYmJmZWNhMzI1Y2E2MTY2NjlmNiIsImlhdCI6MTY5OTk0MTMxMiwiZXhwIjoxNzAwODA1MzEyfQ.Ravi

! You can now access the platform's features.

Go to home

Figure 4-Successful Registration


Batch of IFT 458 Fall 2023
Server:
11/13/2023, 10:55:45 PM
Your IP address is: 70.163.216.246
Course #: CS101

Welcome to Book Exchange!

Student Information

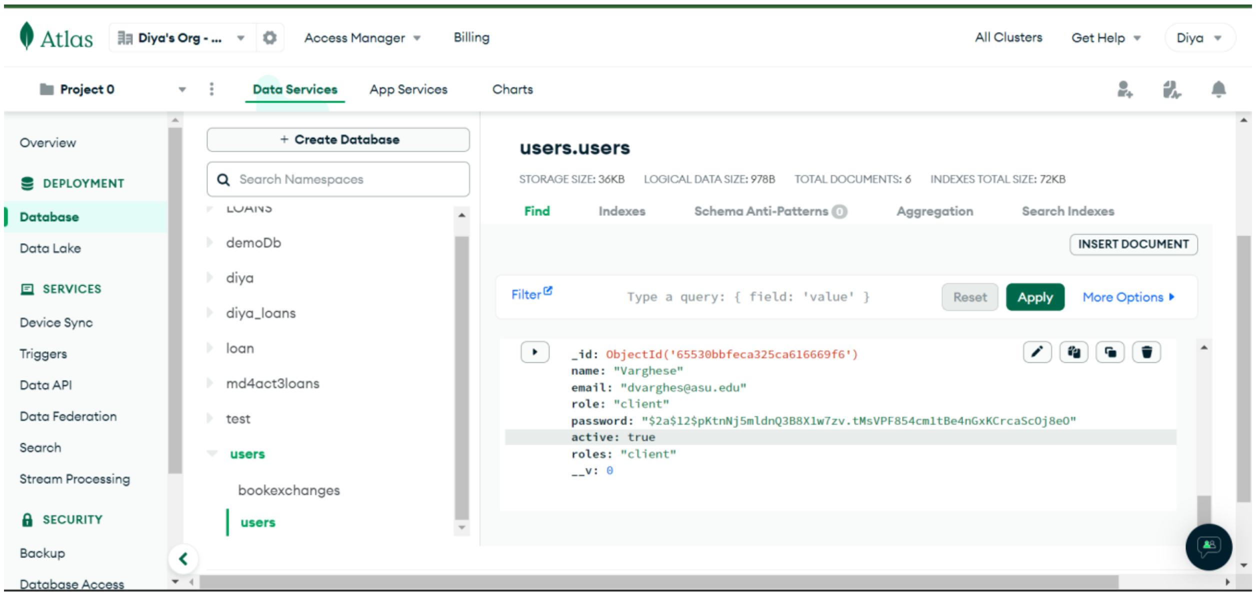
Student Image:

Student Name: Varghese
Student Email: dvarghes@asu.edu
Student ID: 65530bbfeca325ca616669f6
Logout Book Exchange

Available Books:

Book Exchange Platform © IFT 510 - Fall 2023

Figure 5-Go to Home page or Login Page when there is an active JWT Token

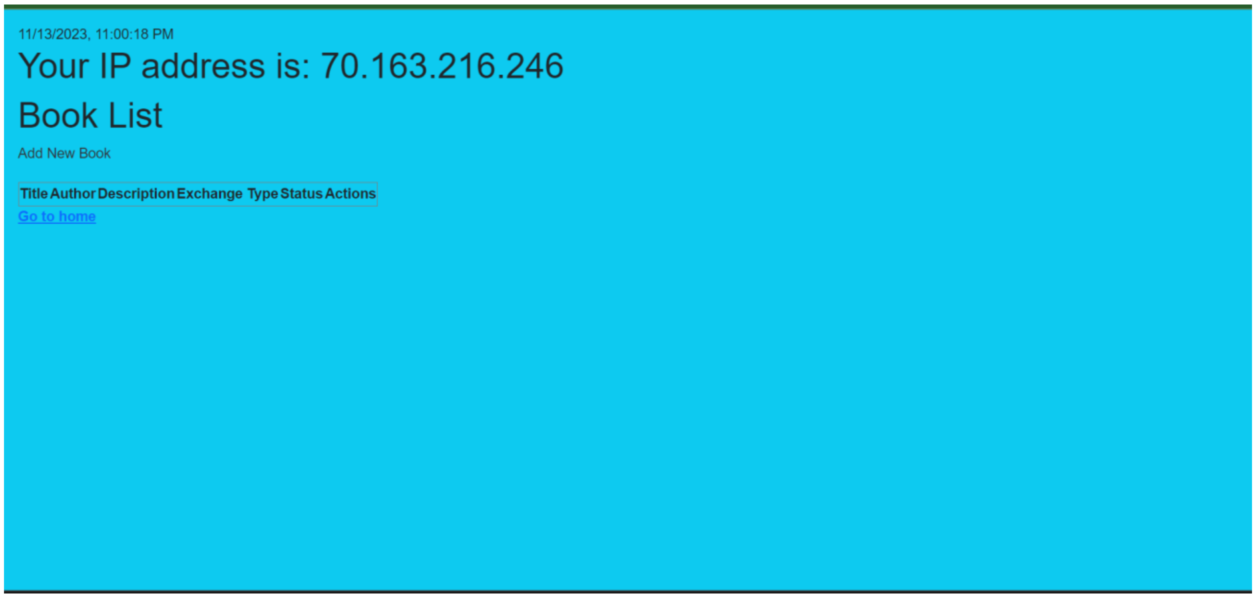


users.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 978B TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 72KB

```
_id: ObjectId('65530bbfeca325ca616669f6')
name: "Varghese"
email: "dvarghes@asu.edu"
role: "client"
password: "$2a$12$pKtnNj5mldnQ3B8X1w7zv.tMsVPF854cm1tBe4nGxKCrcaSc0j8e0"
active: true
roles: "client"
__v: 0
```

Figure 6- User created in the MongoDB - users/users



11/13/2023, 11:00:18 PM

Your IP address is: 70.163.216.246

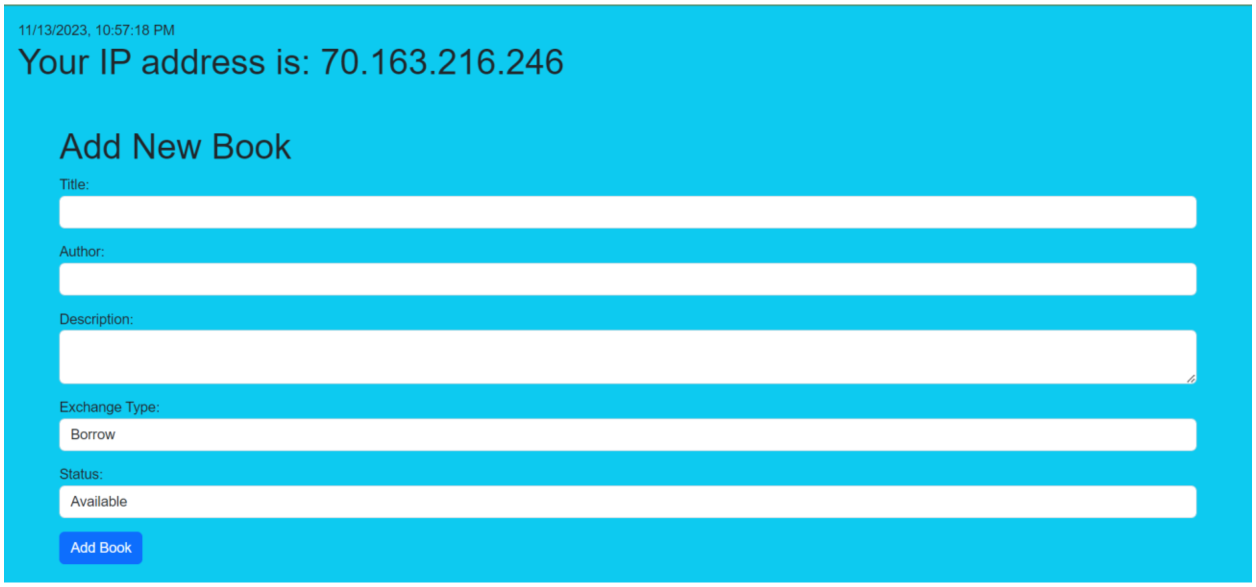
Book List

Add New Book

Title	Author	Description	Exchange Type	Status	Actions

Go to home

Figure 7- BookExchange when there are no books in the DB



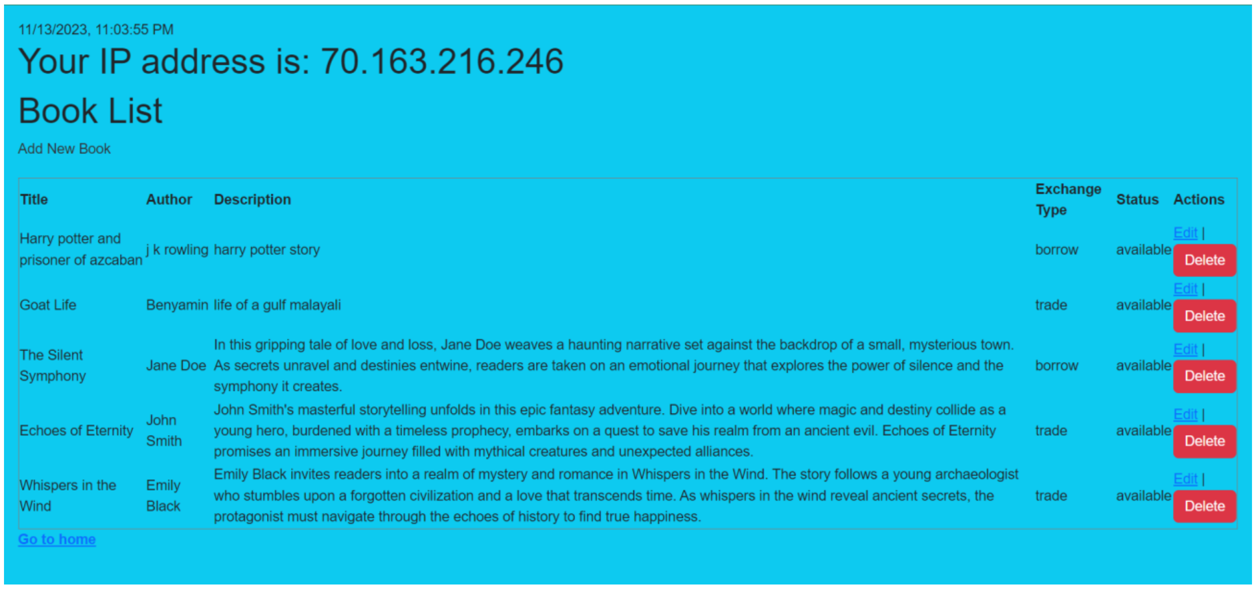
11/13/2023, 10:57:18 PM

Your IP address is: 70.163.216.246

Add New Book

Title:	
Author:	
Description:	
Exchange Type:	Borrow
Status:	Available

Figure 8-Add Book Form



11/13/2023, 11:03:55 PM

Your IP address is: 70.163.216.246

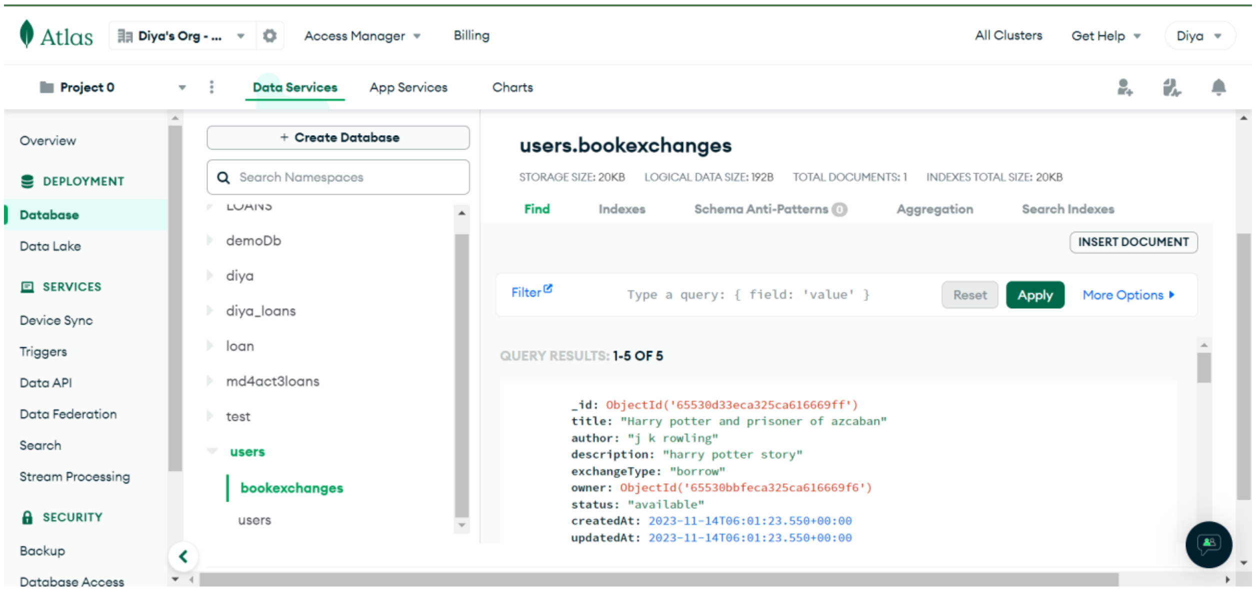
Book List

Add New Book

Title	Author	Description	Exchange Type	Status	Actions
Harry potter and prisoner of azcaban	j k rowling	harry potter story	borrow	available	Edit | Delete
Goat Life	Benyamin	life of a gulf malayali	trade	available	Edit | Delete
The Silent Symphony	Jane Doe	In this gripping tale of love and loss, Jane Doe weaves a haunting narrative set against the backdrop of a small, mysterious town. As secrets unravel and destinies entwine, readers are taken on an emotional journey that explores the power of silence and the symphony it creates.	borrow	available	Edit | Delete
Echoes of Eternity	John Smith	John Smith's masterful storytelling unfolds in this epic fantasy adventure. Dive into a world where magic and destiny collide as a young hero, burdened with a timeless prophecy, embarks on a quest to save his realm from an ancient evil. Echoes of Eternity promises an immersive journey filled with mythical creatures and unexpected alliances.	trade	available	Edit | Delete
Whispers in the Wind	Emily Black	Emily Black invites readers into a realm of mystery and romance in Whispers in the Wind. The story follows a young archaeologist who stumbles upon a forgotten civilization and a love that transcends time. As whispers in the wind reveal ancient secrets, the protagonist must navigate through the echoes of history to find true happiness.	trade	available	Edit | Delete

Go to home

Figure 9-Upon adding 5 different books in the BookExchange page



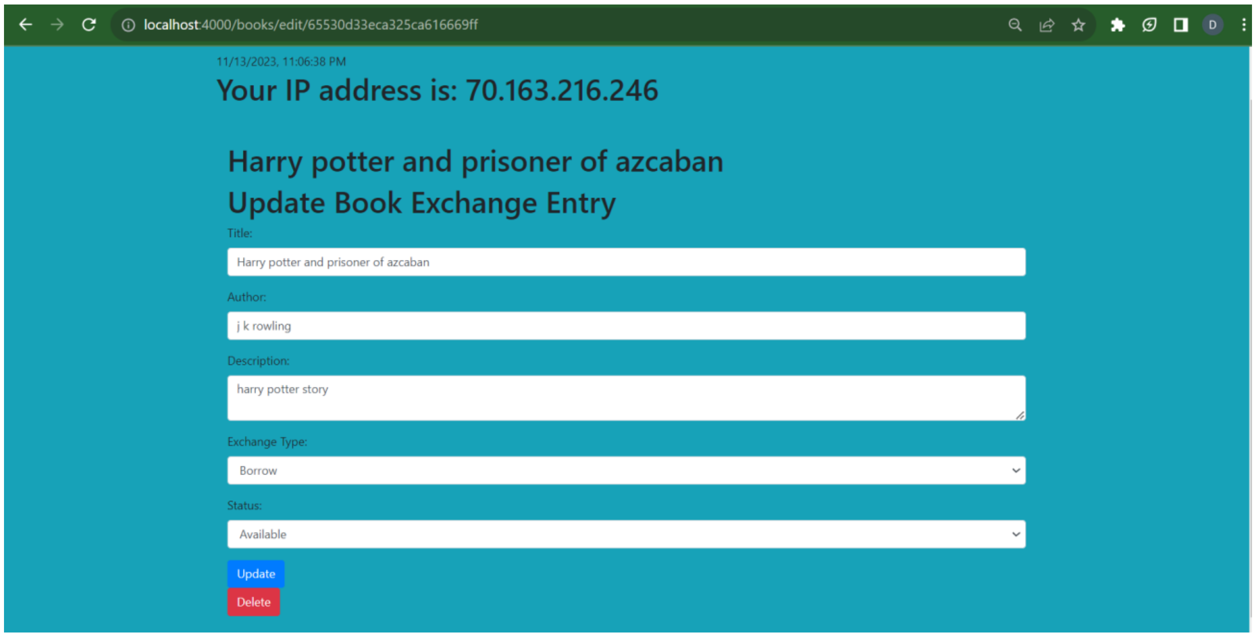
users.bookexchanges

STORAGE SIZE: 20KB LOGICAL DATA SIZE: 192B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 20KB

QUERY RESULTS: 1-5 OF 5

```
_id: ObjectId('65530d33eca325ca616669ff')
title: "Harry potter and prisoner of azcaban"
author: "j k rowling"
description: "harry potter story"
exchangeType: "borrow"
owner: ObjectId('65530bbfeca325ca616669f6')
status: "available"
createdAt: 2023-11-14T06:01:23.550+00:00
updatedAt: 2023-11-14T06:01:23.550+00:00
```

Figure 10-MongoDB upon adding 5 new objects in users/bookexchanges



11/13/2023, 11:06:38 PM

Your IP address is: 70.163.216.246

Harry potter and prisoner of azcaban

Update Book Exchange Entry

Title: Harry potter and prisoner of azcaban

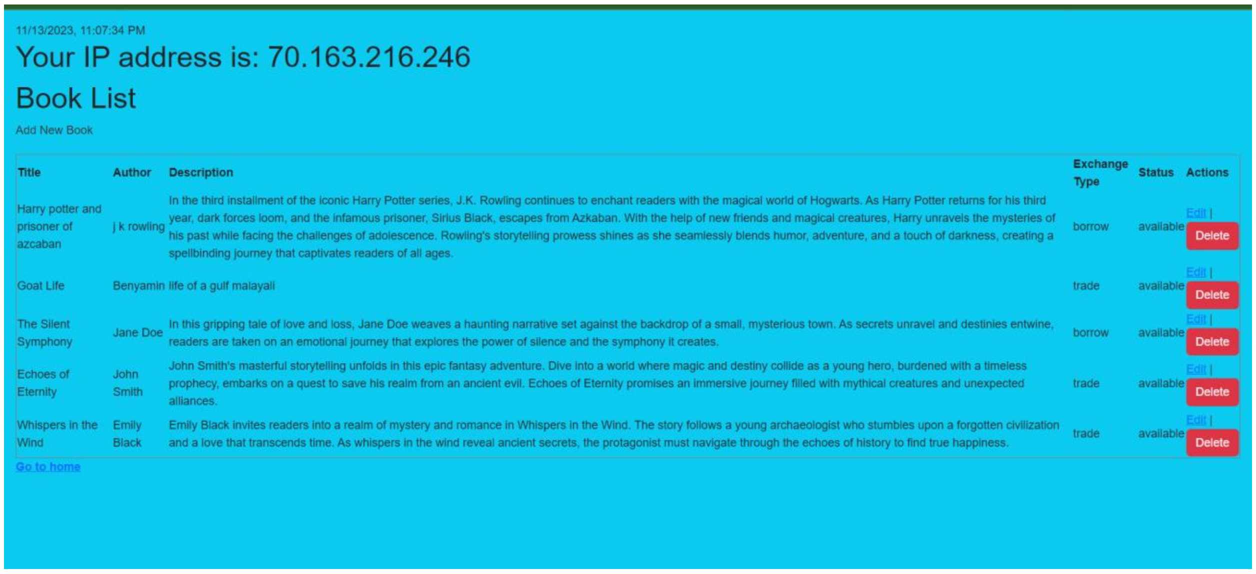
Author: j k rowling

Description: harry potter story

Exchange Type: Borrow

Status: Available

Figure 11-Make changes or update to one of the books


11/13/2023, 11:07:34 PM

Your IP address is: 70.163.216.246

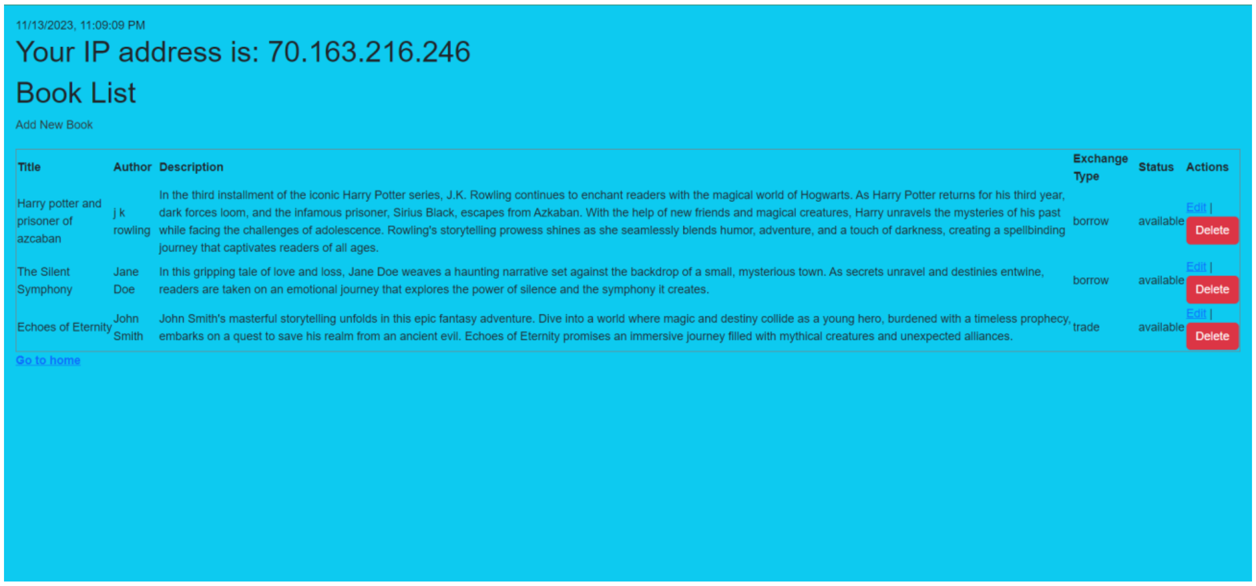
Book List

Add New Book

Title	Author	Description	Exchange Type	Status	Actions
Harry potter and prisoner of azcaban	j k rowling	In the third installment of the iconic Harry Potter series, J.K. Rowling continues to enchant readers with the magical world of Hogwarts. As Harry Potter returns for his third year, dark forces loom, and the infamous prisoner, Sirius Black, escapes from Azkaban. With the help of new friends and magical creatures, Harry unravels the mysteries of his past while facing the challenges of adolescence. Rowling's storytelling prowess shines as she seamlessly blends humor, adventure, and a touch of darkness, creating a spellbinding journey that captivates readers of all ages.	borrow	available	Edit | Delete
Goat Life	Benyamin	life of a gulf malayali	trade	available	Edit | Delete
The Silent Symphony	Jane Doe	In this gripping tale of love and loss, Jane Doe weaves a haunting narrative set against the backdrop of a small, mysterious town. As secrets unravel and destinies entwine, readers are taken on an emotional journey that explores the power of silence and the symphony it creates.	borrow	available	Edit | Delete
Echoes of Eternity	John Smith	John Smith's masterful storytelling unfolds in this epic fantasy adventure. Dive into a world where magic and destiny collide as a young hero, burdened with a timeless prophecy, embarks on a quest to save his realm from an ancient evil. Echoes of Eternity promises an immersive journey filled with mythical creatures and unexpected alliances.	trade	available	Edit | Delete
Whispers in the Wind	Emily Black	Emily Black invites readers into a realm of mystery and romance in Whispers in the Wind. The story follows a young archaeologist who stumbles upon a forgotten civilization and a love that transcends time. As whispers in the wind reveal ancient secrets, the protagonist must navigate through the echoes of history to find true happiness.	trade	available	Edit | Delete

Go to home

Figure 12-After making changes


11/13/2023, 11:09:09 PM

Your IP address is: 70.163.216.246

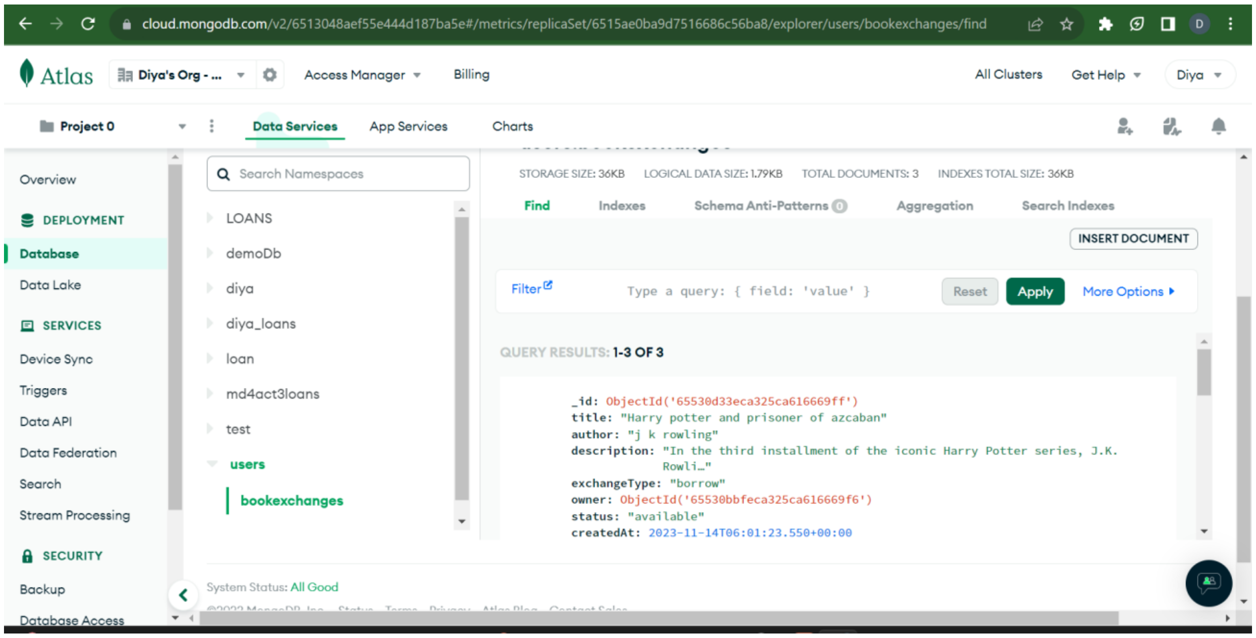
Book List

Add New Book

Title	Author	Description	Exchange Type	Status	Actions
Harry potter and prisoner of azcaban	j k rowling	In the third installment of the iconic Harry Potter series, J.K. Rowling continues to enchant readers with the magical world of Hogwarts. As Harry Potter returns for his third year, dark forces loom, and the infamous prisoner, Sirius Black, escapes from Azkaban. With the help of new friends and magical creatures, Harry unravels the mysteries of his past while facing the challenges of adolescence. Rowling's storytelling prowess shines as she seamlessly blends humor, adventure, and a touch of darkness, creating a spellbinding journey that captivates readers of all ages.	borrow	available	Edit | Delete
The Silent Symphony	Jane Doe	In this gripping tale of love and loss, Jane Doe weaves a haunting narrative set against the backdrop of a small, mysterious town. As secrets unravel and destinies entwine, readers are taken on an emotional journey that explores the power of silence and the symphony it creates.	borrow	available	Edit | Delete
Echoes of Eternity	John Smith	John Smith's masterful storytelling unfolds in this epic fantasy adventure. Dive into a world where magic and destiny collide as a young hero, burdened with a timeless prophecy, embarks on a quest to save his realm from an ancient evil. Echoes of Eternity promises an immersive journey filled with mythical creatures and unexpected alliances.	trade	available	Edit | Delete

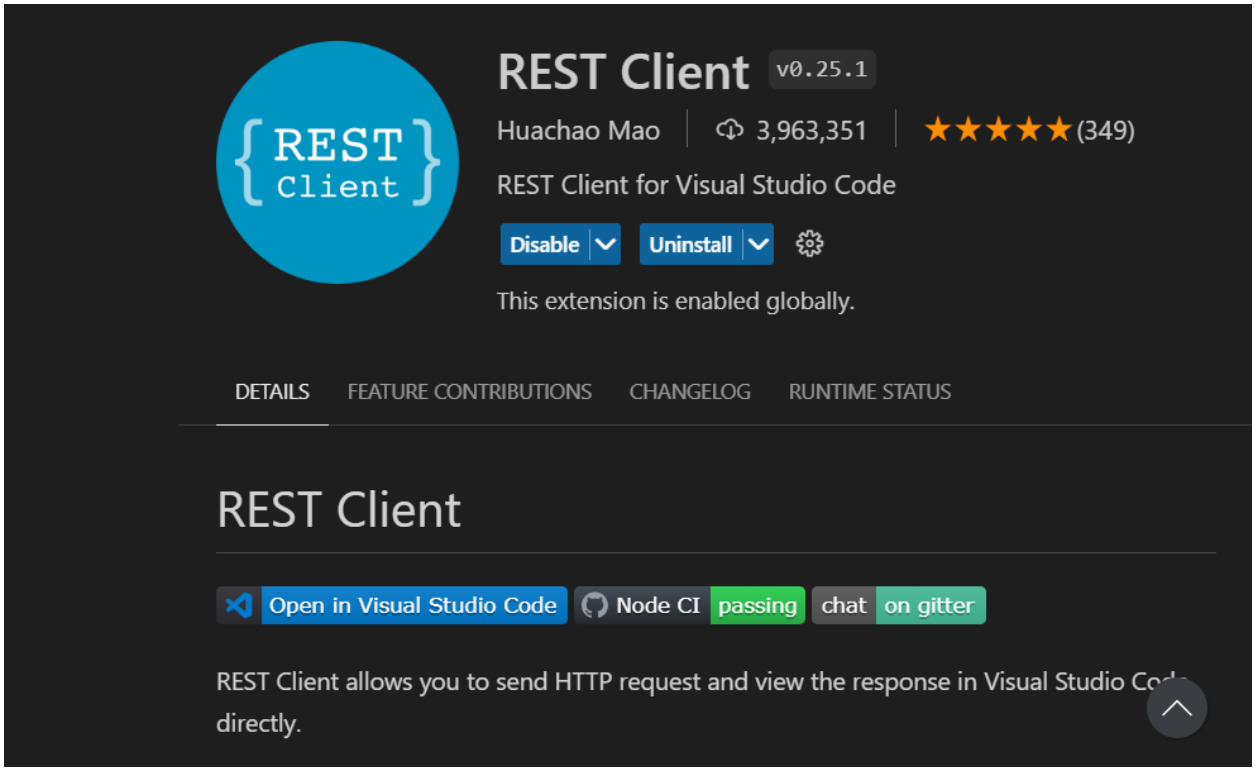
Go to home

Figure 13- Delete a book by clicking on Delete button in the actions column



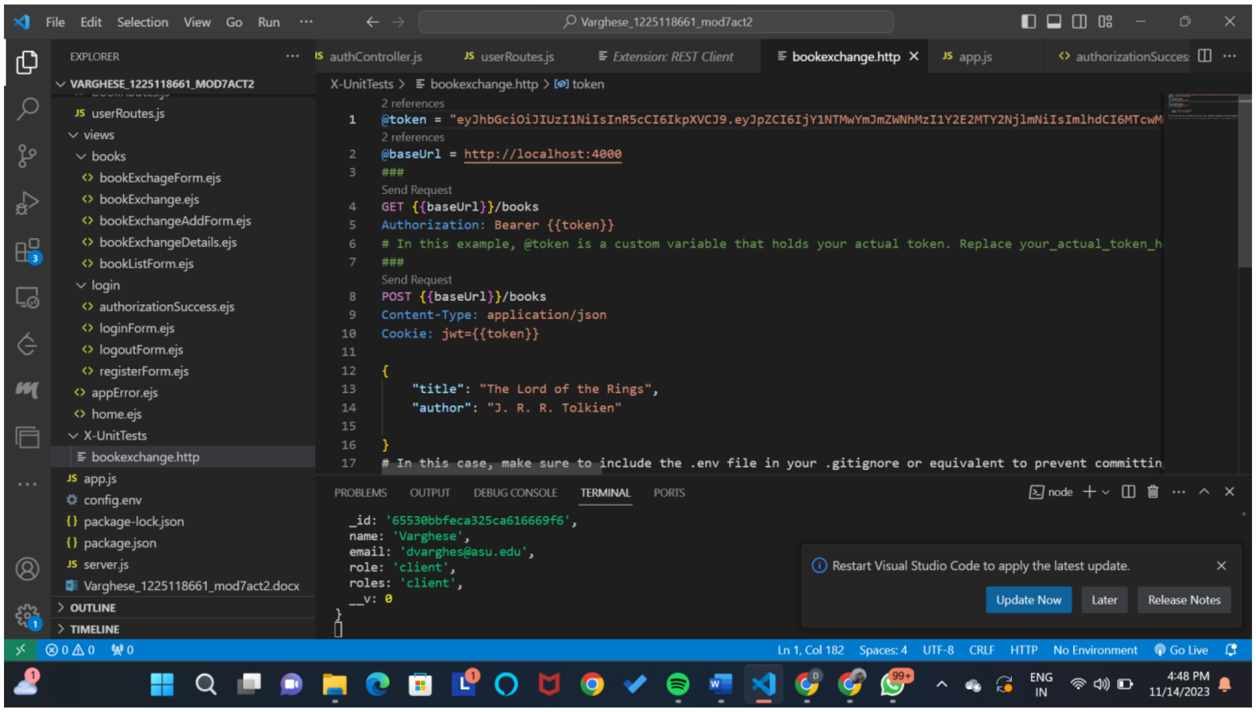
```
_id: ObjectId('65530d33eca325ca616669ff')
title: "Harry potter and prisoner of azcaban"
author: "j k rowling"
description: "In the third installment of the iconic Harry Potter series, J.K.
              Rowli…"
exchangeType: "borrow"
owner: ObjectId('65530bbfeca325ca616669f6')
status: "available"
createdAt: 2023-11-14T06:01:23.550+00:00
```

Figure 14- Deleting reflected in MongoDB



REST Client allows you to send HTTP request and view the response in Visual Studio Code directly.

Figure 15-Tests are done using REST CLIENT

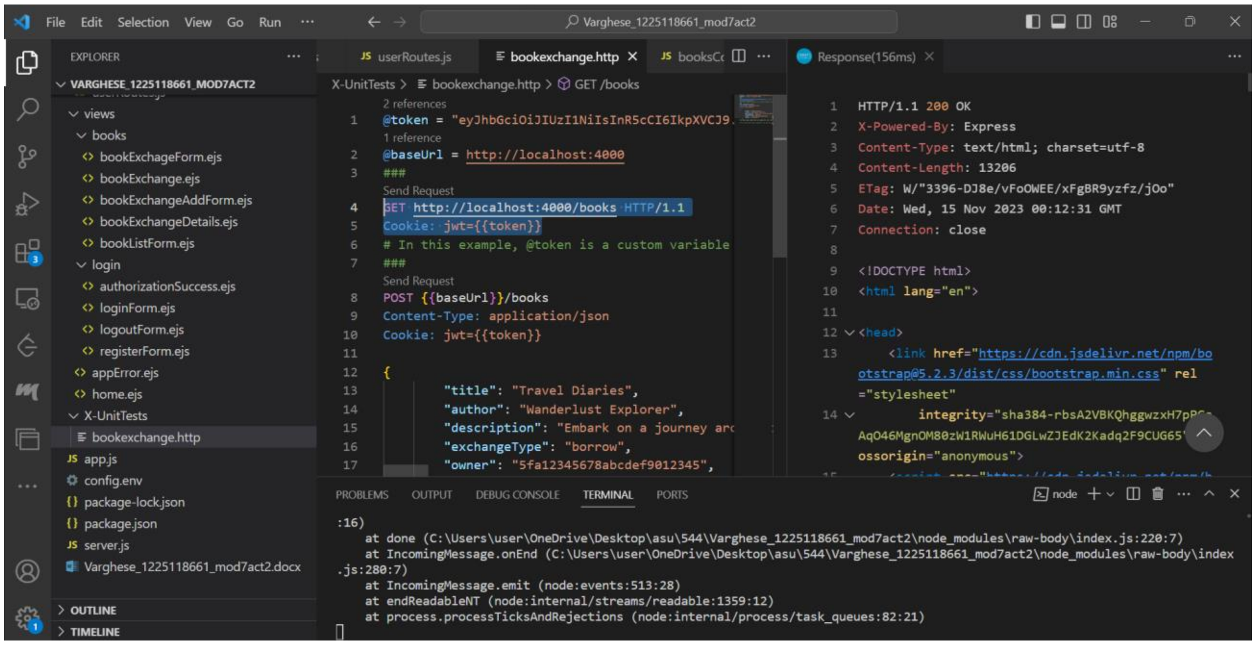


```
@token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1NTMwYmJmZWNhMzI1Y2E2MTY2NjlmNiIsImlhdCI6MTcwM
@baseUrl = http://localhost:4000
###
GET {{baseUrl}}/books
Authorization: Bearer {{token}}
# In this example, @token is a custom variable that holds your actual token. Replace your_actual_token_h
###
POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}

{
    "title": "The Lord of the Rings",
    "author": "J. R. R. Tolkien"

}
# In this case, make sure to include the .env file in your .gitignore or equivalent to prevent committin
```

```
  _id: '65530bbfeca325ca616669f6',
  name: 'Varghese',
  email: 'dvarghes@asu.edu',
  role: 'client',
  roles: 'client',
  __v: 0
}
```

Restart Visual Studio Code to apply the latest update.

Figure 16- Http file for conducting tests

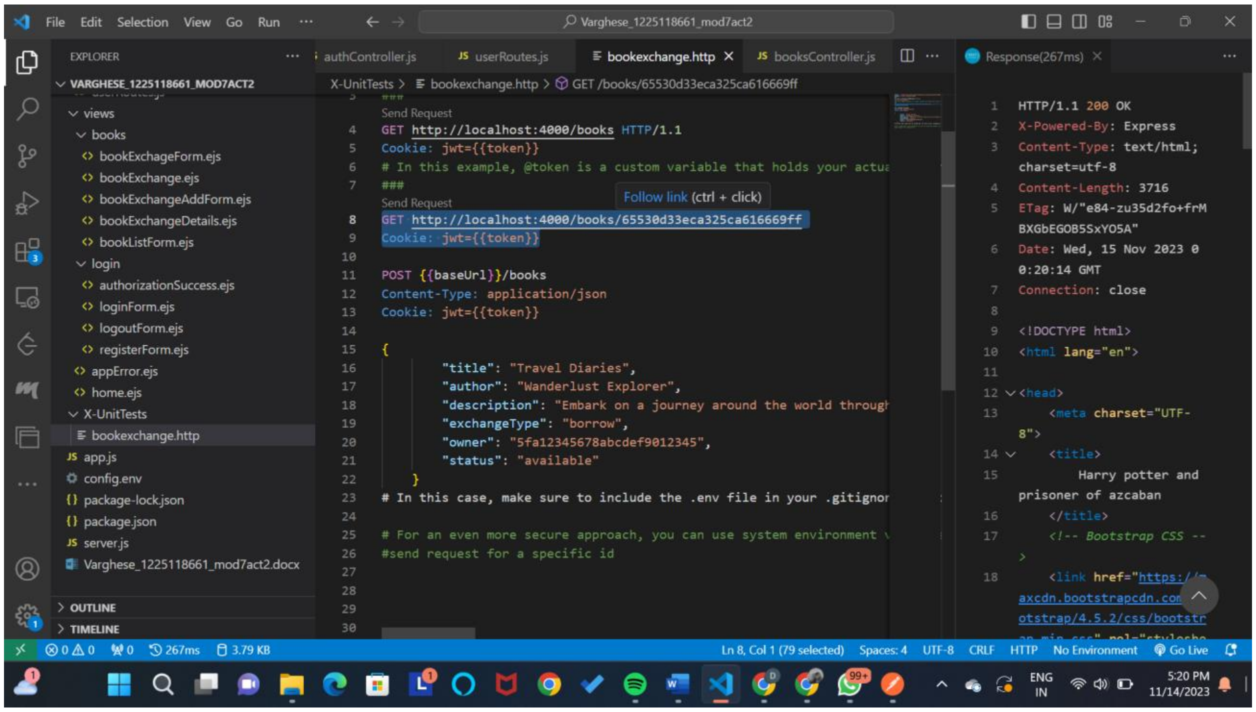


```
@token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
@baseUrl = http://localhost:4000
###
GET http://localhost:4000/books HTTP/1.1
Cookie: jwt={{token}}
# In this example, @token is a custom variable
###
POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}

{
        "title": "Travel Diaries",
        "author": "Wanderlust Explorer",
        "description": "Embark on a journey arc
        "exchangeType": "borrow",
        "owner": "5fa12345678abcdef9012345",
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 13206
ETag: W/"3396-DJ8e/vFoOWEE/xFgBR9yzfz/jOo"
Date: Wed, 15 Nov 2023 00:12:31 GMT
Connection: close

<!DOCTYPE html>
<html lang="en">

<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel
="stylesheet"
        integrity="sha384-rbsA2VBKQhggwzxH7pP
AqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65
ossorigin="anonymous">
```

```
:16)
    at done (C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2\node_modules\raw-body\index.js:220:7)
    at IncomingMessage.onEnd (C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2\node_modules\raw-body\index
.js:280:7)
    at IncomingMessage.emit (node:events:513:28)
    at endReadableNT (node:internal/streams/readable:1359:12)
    at process.processTicksAndRejections (node:internal/process/task_queues:82:21)
```

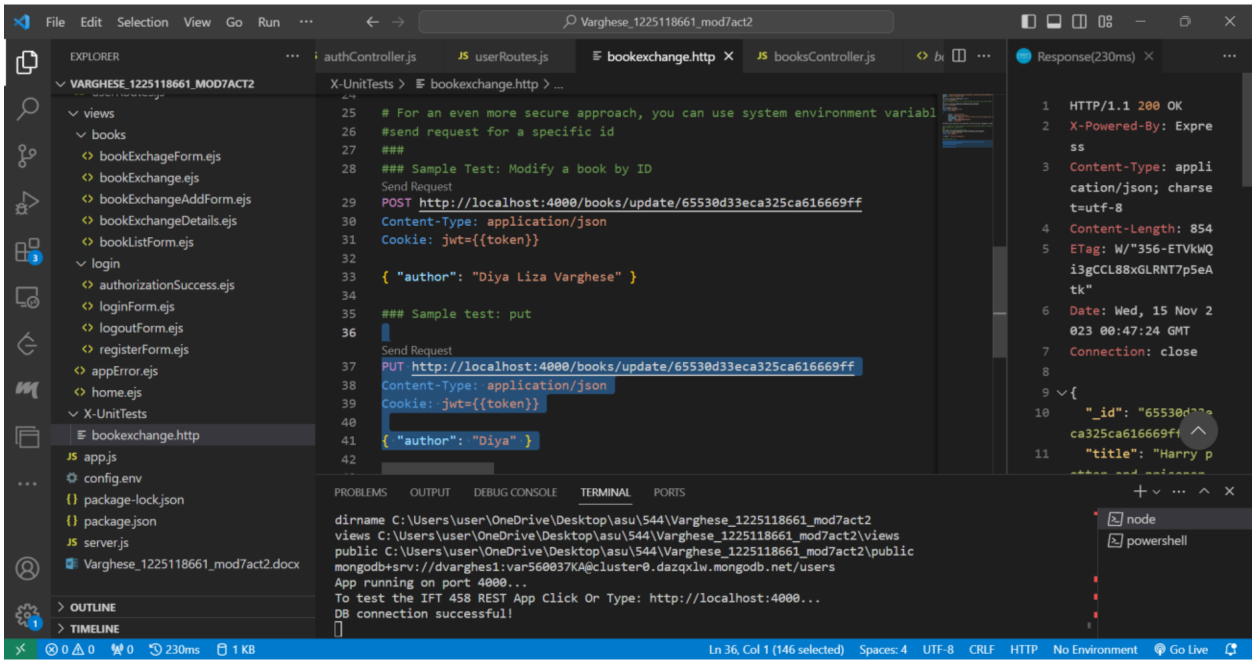
Figure 17-Testing GET request and response is on the right pane



```
###
Send Request
GET http://localhost:4000/books HTTP/1.1
Cookie: jwt={{token}}
# In this example, @token is a custom variable that holds your actua
###
Send Request
GET http://localhost:4000/books/65530d33eca325ca616669ff
Cookie: jwt={{token}}

POST {{baseUrl}}/books
Content-Type: application/json
Cookie: jwt={{token}}

{
        "title": "Travel Diaries",
        "author": "Wanderlust Explorer",
        "description": "Embark on a journey around the world throug
        "exchangeType": "borrow",
        "owner": "5fa12345678abcdef9012345",
        "status": "available"
    }
# In this case, make sure to include the .env file in your .gitignor

# For an even more secure approach, you can use system environment 
#send request for a specific id
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 3716
ETag: W/"e84-zu35d2fo+frMBXGbEGOB5SxYO5A"
Date: Wed, 15 Nov 2023 00:20:14 GMT
Connection: close

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>
        Harry potter and prisoner of azcaban
    </title>
    <!-- Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstr
```

Figure 18-GET http://localhost:4000/books/65530d33eca325ca616669ff

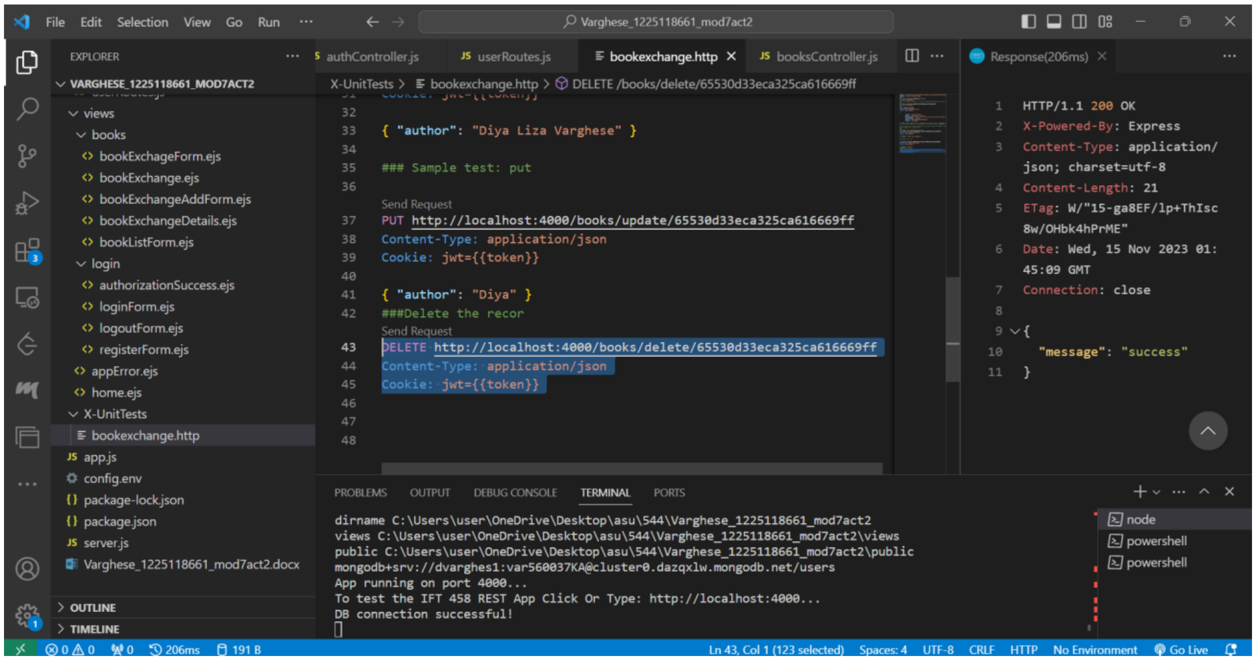


```
# For an even more secure approach, you can use system environment variabl
#send request for a specific id
###
### Sample Test: Modify a book by ID
Send Request
POST http://localhost:4000/books/update/65530d33eca325ca616669ff
Content-Type: application/json
Cookie: jwt={{token}}

{ "author": "Diya Liza Varghese" }

### Sample test: put

Send Request
PUT http://localhost:4000/books/update/65530d33eca325ca616669ff
Content-Type: application/json
Cookie: jwt={{token}}

{ "author": "Diya" }
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 854
ETag: W/"356-ETVkWQi3gCCL88xGLRNT7p5eAtk"
Date: Wed, 15 Nov 2023 00:47:24 GMT
Connection: close

{
  "_id": "65530d33eca325ca616669ff",
  "title": "Harry p
```

```
dirname C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2
views C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2\views
public C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2\public
mongodb+srv://dvarghes1:var560037KA@cluster0.dazqxlw.mongodb.net/users
App running on port 4000...
To test the IFT 458 REST App Click Or Type: http://localhost:4000...
DB connection successful!
```

Figure 19-6.PUT Request: Modify a Specific Book by ID:

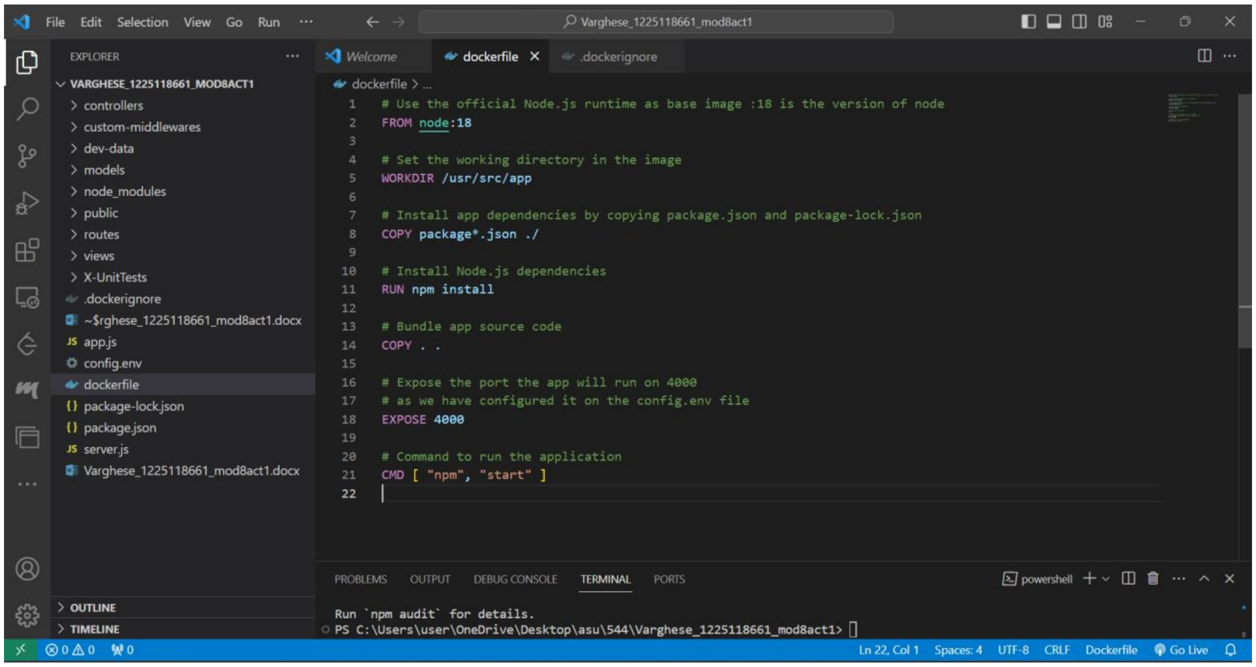


```
{ "author": "Diya Liza Varghese" }

### Sample test: put

PUT http://localhost:4000/books/update/65530d33eca325ca616669ff
Content-Type: application/json
Cookie: jwt={{token}}

{ "author": "Diya" }
###Delete the recor
DELETE http://localhost:4000/books/delete/65530d33eca325ca616669ff
Content-Type: application/json
Cookie: jwt={{token}}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 21
ETag: W/"15-ga8EF/lp+ThIsc8w/OHbk4hPrME"
Date: Wed, 15 Nov 2023 01:45:09 GMT
Connection: close

{
  "message": "success"
}
```

```
dirname C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2
views C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2\views
public C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod7act2\public
mongodb+srv://dvarghes1:var560037KA@cluster0.dazqxlw.mongodb.net/users
App running on port 4000...
To test the IFT 458 REST App Click Or Type: http://localhost:4000...
DB connection successful!
```

Figure 20-testing:7. DELETE Request: Erase a Specific Book by ID

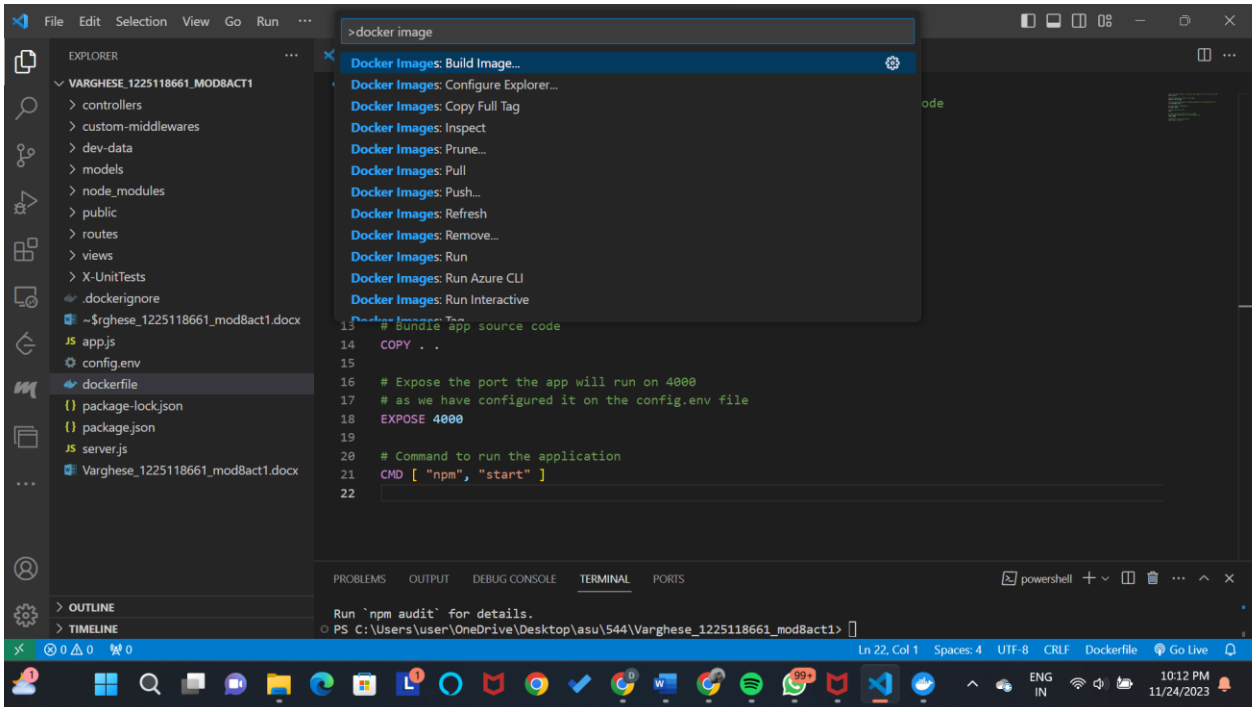


```
# Use the official Node.js runtime as base image :18 is the version of node
FROM node:18

# Set the working directory in the image
WORKDIR /usr/src/app

# Install app dependencies by copying package.json and package-lock.json
COPY package*.json ./

# Install Node.js dependencies
RUN npm install

# Bundle app source code
COPY . .

# Expose the port the app will run on 4000
# as we have configured it on the config.env file
EXPOSE 4000

# Command to run the application
CMD [ "npm", "start" ]
```

```
Run `npm audit` for details.
PS C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod8act1>
```

Figure 21-2. Creating dockerfile

Docker image Build:

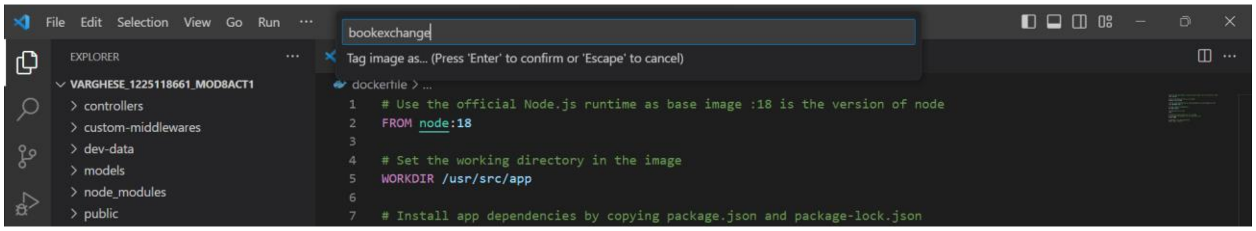


```
>docker image
Docker Images: Build Image...
Docker Images: Configure Explorer...
Docker Images: Copy Full Tag
Docker Images: Inspect
Docker Images: Prune...
Docker Images: Pull
Docker Images: Push...
Docker Images: Refresh
Docker Images: Remove...
Docker Images: Run
Docker Images: Run Azure CLI
Docker Images: Run Interactive
```

```
# Bundle app source code
COPY . .

# Expose the port the app will run on 4000
# as we have configured it on the config.env file
EXPOSE 4000

# Command to run the application
CMD [ "npm", "start" ]
```

```
Run `npm audit` for details.
PS C:\Users\user\OneDrive\Desktop\asu\544\Varghese_1225118661_mod8act1>
```

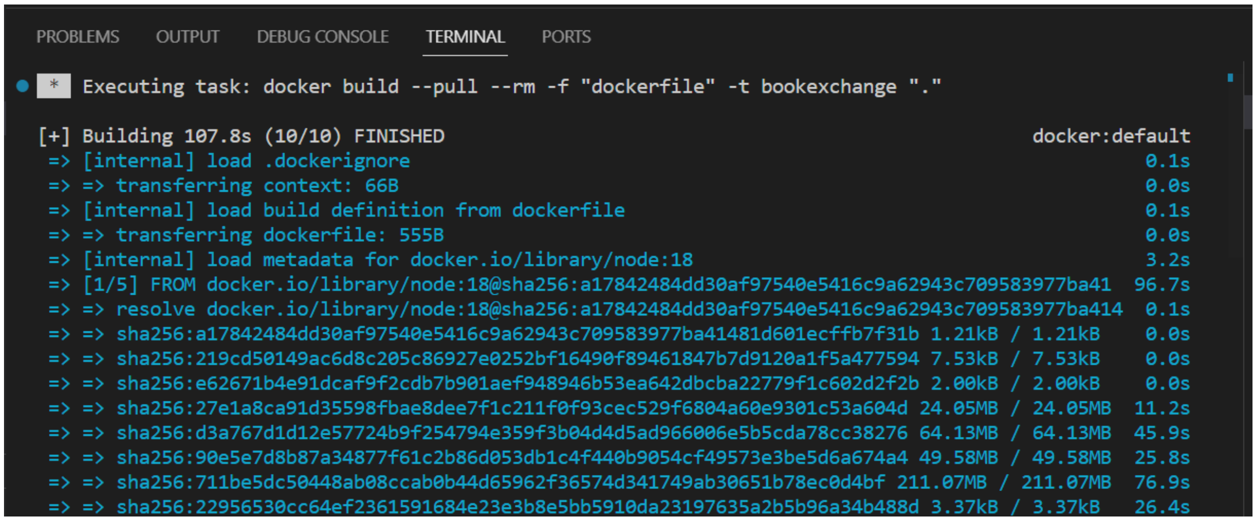
Figure 22-docker image build 1



```
bookexchange
Tag image as... (Press 'Enter' to confirm or 'Escape' to cancel)
```

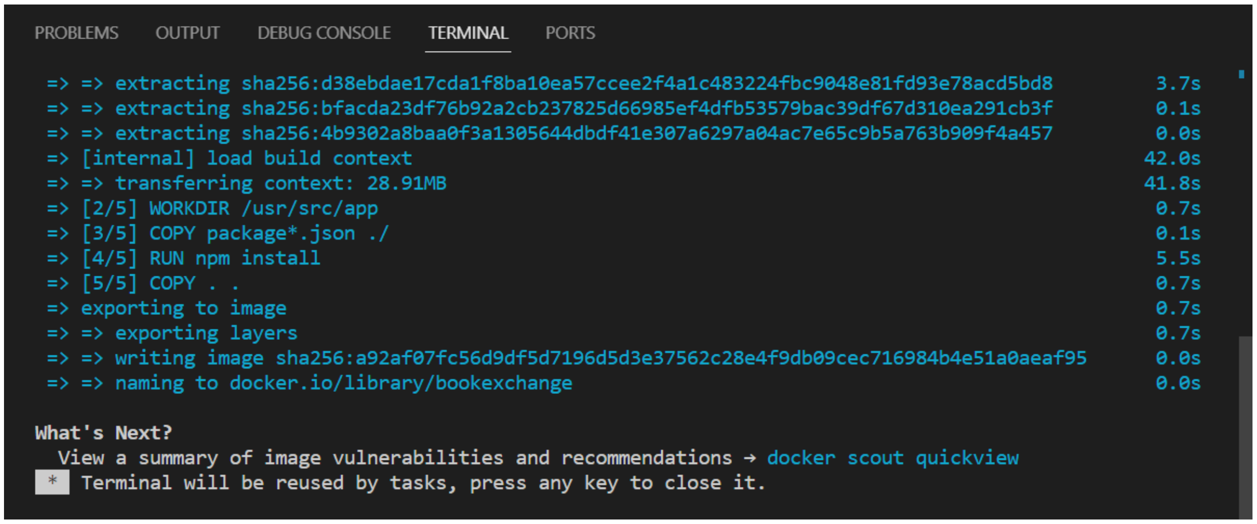
```
# Use the official Node.js runtime as base image :18 is the version of node
FROM node:18

# Set the working directory in the image
WORKDIR /usr/src/app

# Install app dependencies by copying package.json and package-lock.json
```

Figure 23-docker image build 2



```
 * Executing task: docker build --pull --rm -f "dockerfile" -t bookexchange "."

[+] Building 107.8s (10/10) FINISHED                                                         docker:default
 => [internal] load .dockerignore                                                                     0.1s
 => => transferring context: 66B                                                                      0.0s
 => [internal] load build definition from dockerfile                                                  0.1s
 => => transferring dockerfile: 555B                                                                  0.0s
 => [internal] load metadata for docker.io/library/node:18                                            3.2s
 => [1/5] FROM docker.io/library/node:18@sha256:a17842484dd30af97540e5416c9a62943c709583977ba41       96.7s
 => => resolve docker.io/library/node:18@sha256:a17842484dd30af97540e5416c9a62943c709583977ba414      0.1s
 => => sha256:a17842484dd30af97540e5416c9a62943c709583977ba41481d601ecffb7f31b 1.21kB / 1.21kB         0.0s
 => => sha256:219cd50149ac6d8c205c86927e0252bf16490f89461847b7d9120a1f5a477594 7.53kB / 7.53kB         0.0s
 => => sha256:e62671b4e91dcaf9f2cdb7b901aef948946b53ea642dbcba22779f1c602d2f2b 2.00kB / 2.00kB         0.0s
 => => sha256:27e1a8ca91d35598fbae8dee7f1c211f0f93cec529f6804a60e9301c53a604d 24.05MB / 24.05MB       11.2s
 => => sha256:d3a767d1d12e57724b9f254794e359f3b04d4d5ad966006e5b5cda78cc38276 64.13MB / 64.13MB       45.9s
 => => sha256:90e5e7d8b87a34877f61c2b86d053db1c4f440b9054cf49573e3be5d6a674a4 49.58MB / 49.58MB       25.8s
 => => sha256:711be5dc50448ab08ccab0b44d65962f36574d341749ab30651b78ec0d4bf 211.07MB / 211.07MB       76.9s
 => => sha256:22956530cc64ef2361591684e23e3b8e5bb5910da23197635a2b5b96a34b488d 3.37kB / 3.37kB        26.4s
```

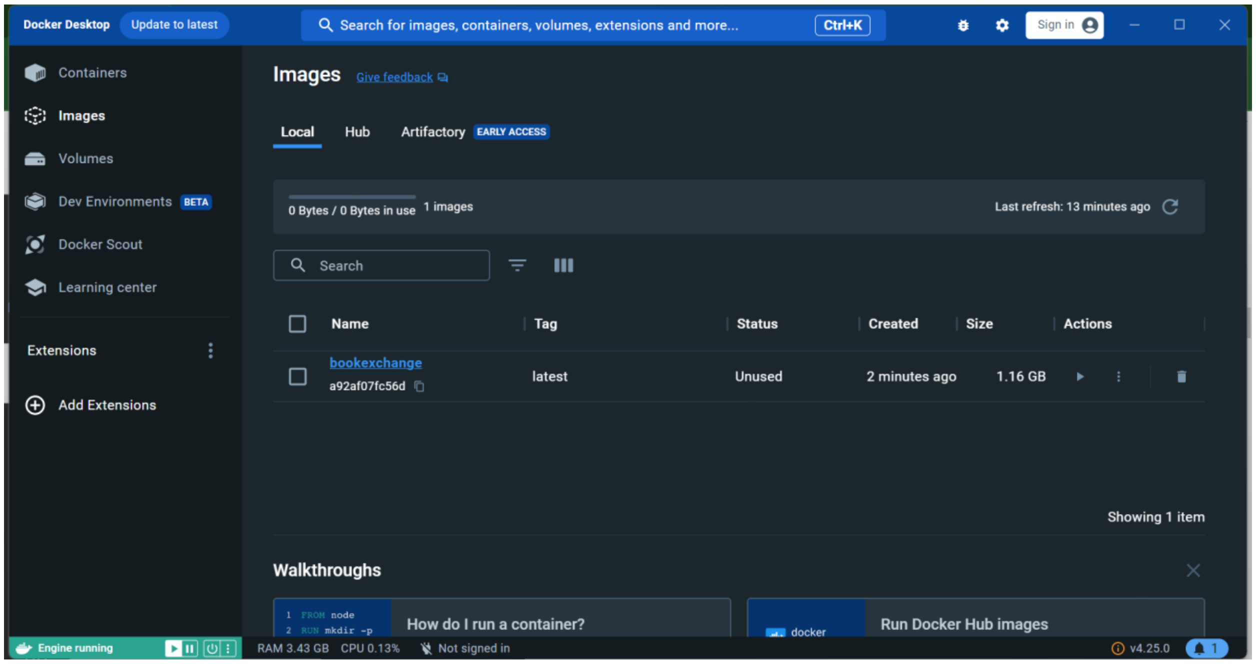
Figure 24-docker image build 3



```
 => => extracting sha256:d38ebdae17cda1f8ba10ea57ccee2f4a1c483224fbc9048e81fd93e78acd5bd8         3.7s
 => => extracting sha256:bfacda23df76b92a2cb237825d66985ef4dfb53579bac39df67d310ea291cb3f         0.1s
 => => extracting sha256:4b9302a8baa0f3a1305644dbdf41e307a6297a04ac7e65c9b5a763b909f4a457         0.0s
 => [internal] load build context                                                                42.0s
 => => transferring context: 28.91MB                                                             41.8s
 => [2/5] WORKDIR /usr/src/app                                                                    0.7s
 => [3/5] COPY package*.json ./                                                                   0.1s
 => [4/5] RUN npm install                                                                         5.5s
 => [5/5] COPY . .                                                                                0.7s
 => exporting to image                                                                            0.7s
 => => exporting layers                                                                           0.7s
 => => writing image sha256:a92af07fc56d9df5d7196d5d3e37562c28e4f9db09cec716984b4e51a0aeaf95      0.0s
 => => naming to docker.io/library/bookexchange                                                   0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
 * Terminal will be reused by tasks, press any key to close it.
```

Figure 25-docker image build 4

Run the Docker Container:



Images Give feedback

Local | Hub | Artifactory EARLY ACCESS

0 Bytes / 0 Bytes in use 1 images Last refresh: 13 minutes ago

Name	Tag	Status	Created	Size	Actions
bookexchange a92af07fc56d	latest	Unused	2 minutes ago	1.16 GB	

Showing 1 item

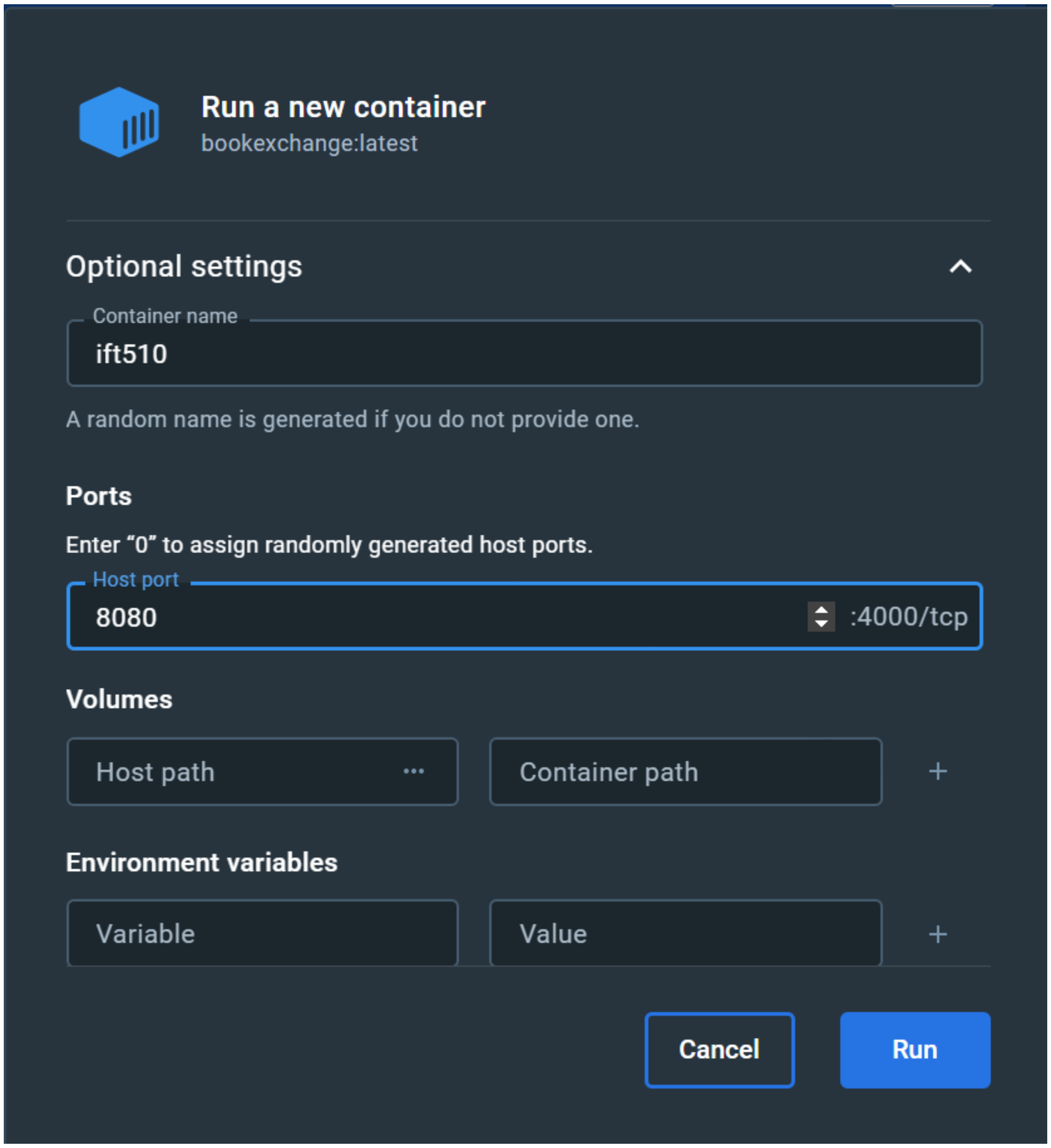
Walkthroughs

How do I run a container?

Run Docker Hub images

Engine running RAM 3.43 GB CPU 0.13% Not signed in v4.25.0

Figure 26-Running the docker container 1



Run a new container
bookexchange:latest

Optional settings

Container name: ift510

A random name is generated if you do not provide one.

Ports

Enter "0" to assign randomly generated host ports.

Host port: 8080 :4000/tcp

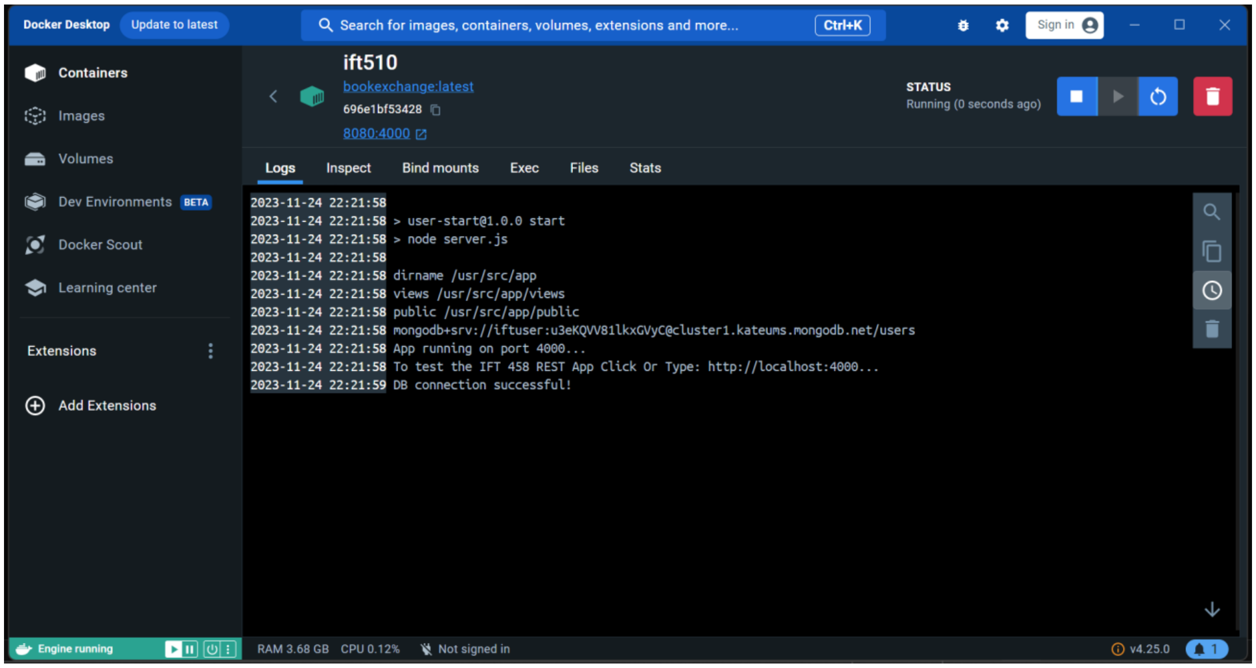
Volumes

Host path | Container path

Environment variables

Variable | Value

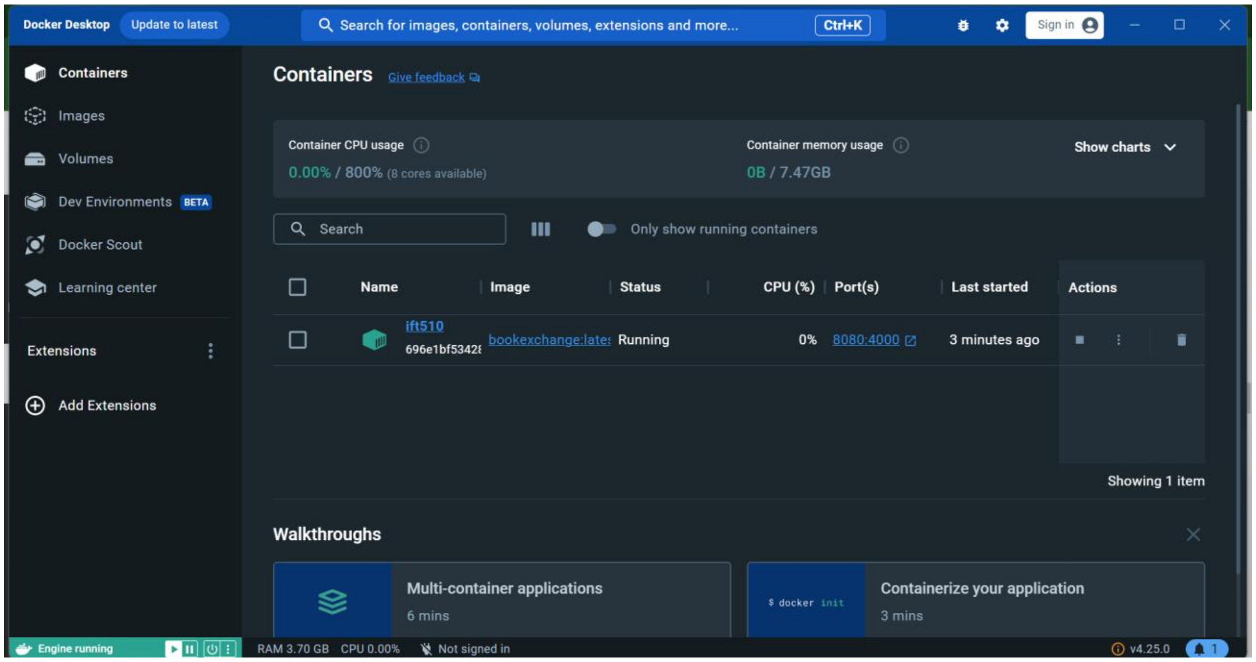
Figure 27-Running the docker container 2


ift510
bookexchange:latest
696e1bf53428
8080:4000

STATUS
Running (0 seconds ago)

```
2023-11-24 22:21:58
2023-11-24 22:21:58 > user-start@1.0.0 start
2023-11-24 22:21:58 > node server.js
2023-11-24 22:21:58
2023-11-24 22:21:58 dirname /usr/src/app
2023-11-24 22:21:58 views /usr/src/app/views
2023-11-24 22:21:58 public /usr/src/app/public
2023-11-24 22:21:58 mongodb+srv://iftuser:u3eKQVV81lkxGVyC@cluster1.kateums.mongodb.net/users
2023-11-24 22:21:58 App running on port 4000...
2023-11-24 22:21:58 To test the IFT 458 REST App Click Or Type: http://localhost:4000...
2023-11-24 22:21:59 DB connection successful!
```

Figure 28-Running the docker container 3

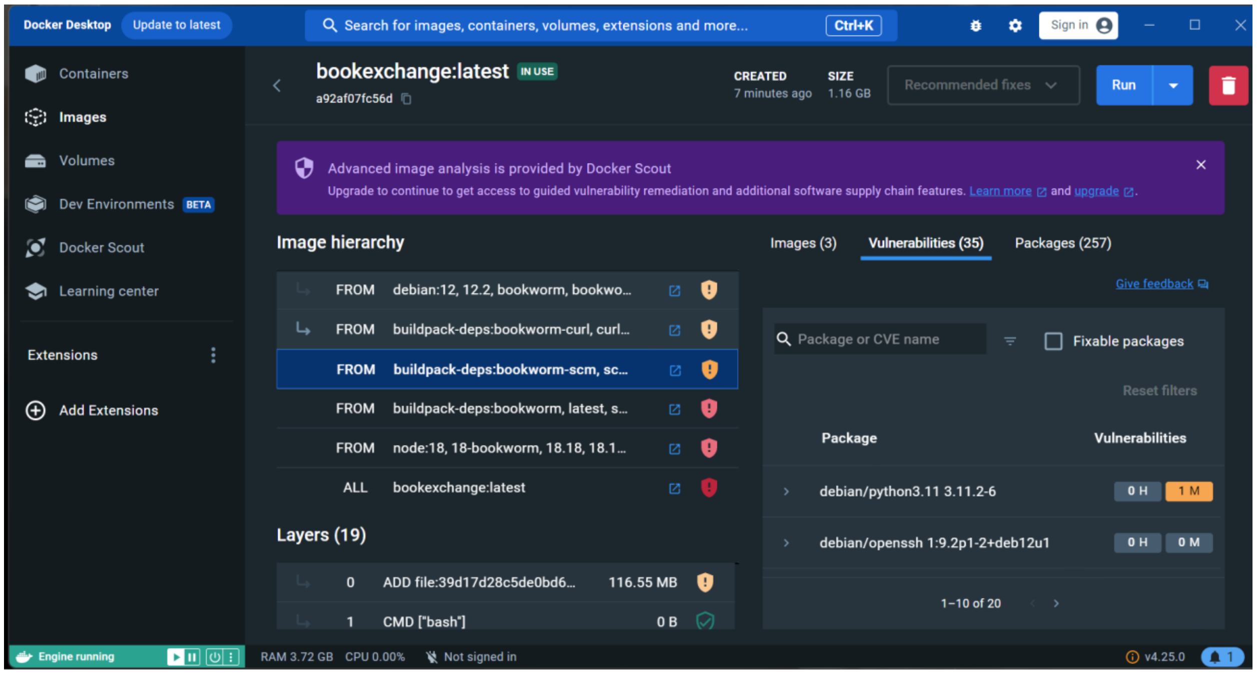

Containers

Container CPU usage: 0.00% / 800% (8 cores available)
Container memory usage: 0B / 7.47GB

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
	ift510 696e1bf5342ε	bookexchange:late:	Running	0%	8080:4000	3 minutes ago	

Showing 1 item

Figure 29-Running the docker container 4



bookexchange:latest IN USE
a92af07fc56d
CREATED 7 minutes ago
SIZE 1.16 GB

Advanced image analysis is provided by Docker Scout
Upgrade to continue to get access to guided vulnerability remediation and additional software supply chain features. Learn more and upgrade.

Image hierarchy

FROM	debian:12, 12.2, bookworm, bookwo...
FROM	buildpack-deps:bookworm-curl, curl...
FROM	buildpack-deps:bookworm-scm, sc...
FROM	buildpack-deps:bookworm, latest, s...
FROM	node:18, 18-bookworm, 18.18, 18.1...
ALL	bookexchange:latest

Layers (19)

0	ADD file:39d17d28c5de0bd6...	116.55 MB
1	CMD ["bash"]	0 B

Images (3) Vulnerabilities (35) Packages (257)

Package or CVE name
Fixable packages

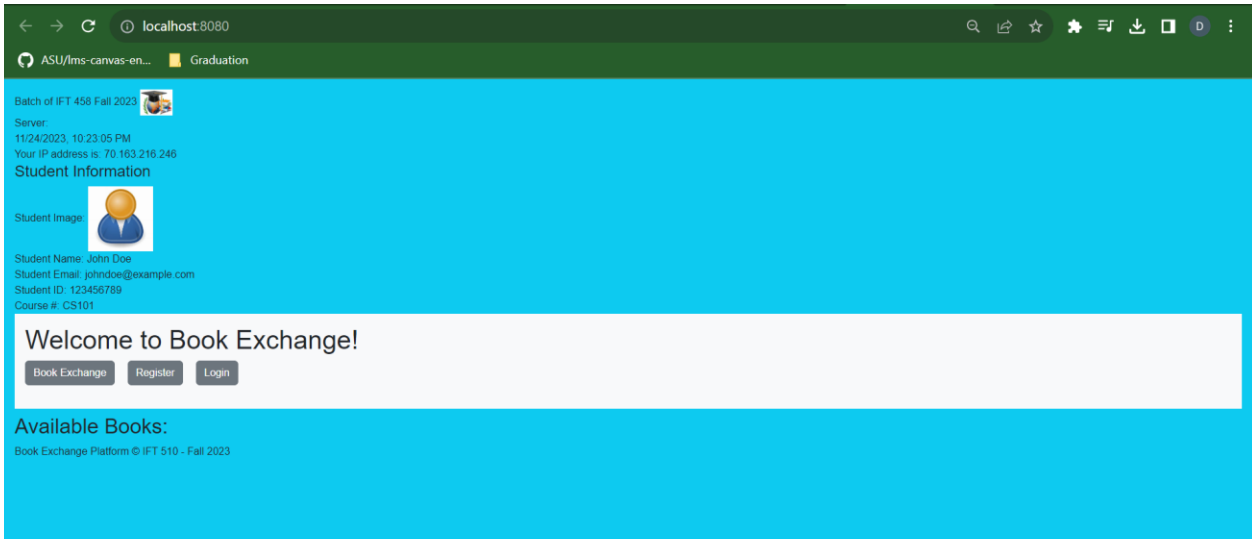
Package	Vulnerabilities
debian/python3.11 3.11.2-6	0 H 1 M
debian/openssh 1:9.2p1-2+deb12u1	0 H 0 M

1–10 of 20

RAM 3.72 GB CPU 0.00% Not signed in v4.25.0

Figure 30-Running the docker container 5

Accessing the Node.js Application:



Batch of IFT 458 Fall 2023
Server:
11/24/2023, 10:23:05 PM
Your IP address is: 70.163.216.246
Student Information
Student Image:
Student Name: John Doe
Student Email: johndoe@example.com
Student ID: 123456789
Course #: CS101

Welcome to Book Exchange!
Book Exchange Register Login

Available Books:
Book Exchange Platform © IFT 510 - Fall 2023

Figure 31-The application that runs on port 8080 upon configuration of docker

IMPLEMENTATION

The project runs on NodeJS, the front end is using EJS and the database support is done by MongoDB. The project uses JWT tokens for authentication. The architecture is based off MVC. The routes are defined in routes folder, then there are controllers, who render the UI using views templates. There are middlewares used in the project – which are global and specific. Authentication middleware, views middleware, body-parser etc.

The server.js is the starter file for the application. It imports app.js which has the major dependencies included. The starter file also has MongoDB connection established using the credentials stored in the config.env. The config file has DB name, DB password etc regarding the DB connection. The app.js file sets the view engine as ejs and sets the Views folder as the directory to pull out the views. The global middlewares (How to Build Middleware for Node.js: A Complete Guide, 2022) that are used are express, cookie-parser and body parser. These middlewares will be passed into all the api calls inspite of being mentioned. The home route is also mentioned which checks the user name if it is present or not, and if it is present, it pulls out the information of the user to be displayed in the home page. The routes for the user, userRoute and books, booksRouter is also mentioned to be used in the app using app.use. This makes the routing working.

In the userRoute, there is routers mentiones which will route the logic when each routes are called eg: /signup, /login, /logout for both get and post. It has different controller functions menationed to be called when each of the router is called. The controller used is authController. When the user clicks on the SignUp button in the home page, it reroutes to GET /users/signup.

This is defined to connect to authcontroller.signup. When login button is clicked, it will render /users/login.

The different functions in authcontroller are as follows:

1. Signup

The function renders the registerForm. If it is a post call, the function takes in name, email, password, and confirmation. It tries to create a token using this information. The createSendToken calls signToken function on the user object. The user object is modelled according to models/userModel/userSchema. It gets posted into the database after encrypting the password. It sets a cookie for the response object with an expiry as well. And it renders an authorizationSuccess page. The signToken function signs the id using JWT and renders the login page.

2. Login

The login page is rendered from the views/login/loginForm. Once the data is inserted in the fields, it will send the email and password to the function. It check sthe email and password is correct, I will render failure pages iof there is an error, otherwise, it will finc the user id in the MongoDB – users database and check the same. It will create a token for the user object and render the authSuccess page.

3. LogOutForm

When a user wants to logout, the function is invoked and it will clear the cookie and send it back with a null object.

Once the login is odne, then the home page with the user details will be pulled up. It gives an opportunity to click on BookExchange button which reroutes to /books. The booksRouter has all the routes that will be useful for this case regarding books. The get / route in booksRouter will use the getBooks method from booksController. It will pass authenticate and authorize which are custom-middlewares in the authMiddleware. We can pass as many middlewares as required in NodeJS. The booksController has the following funcions.

1. getBooks

   It will pull back all the books in the mongodb book model. And it will render books/bookListForm and pass the books list to it and the view page will display that there. If one wants to edit the book, /books/edit/:id is called and it reroutes to updateBookForm.

2. updateBookForm

   this function will take in and load the form with the data of your book that you selected. Upon changing the data there, you can click on submit and it will reroute to /books/update/:id. This will call updateBookById.

3. updateBookById

   The function find the object from DB using he book id and then posts back to the database by taking in the changes from the form. It will re-render the booListForm back.

4. addBook

When you want to add a book, the button in the bookexchangelist, will re-route you to addBook and it renders books/bookExchangeAddForm. There will be a button in the form which will submit the book.

7. createBook

Once the form is filled, it will submit the book object into books database using he books model that is created in the models/bookModel/bookExchangeSchema.

8. deleteBookById

The deletebookById function is called when the delet button in the page is clicked and the user Id is passed, it will remove the object from the database.

There is the custom middleware, which is the authmiddleware which is used by the books routes, whichever method is called there. This middleware has 2 methods – authenticate and authorize. These 2 methods will check if the user logged in authenticated to use the application. It makes sure the token is active. Authorize method checks if this user is present in the Database.

In addition to the robust implementation described earlier, noteworthy enhancements have been made to the project. Unit testing has been diligently conducted using the REST Client extension in Visual Studio Code, ensuring that critical functions and components are thoroughly validated. This systematic testing approach (Waeosri, 2022) contributes to the reliability and stability of the application, identifying and addressing potential issues proactively. Furthermore, the project has been containerized using Docker, leveraging its containerization capabilities to encapsulate the application and its dependencies into a portable and reproducible container. Docker not only facilitates seamless deployment across various environments but also streamlines the scaling process by enabling the efficient creation of multiple container instances. These technological additions align with best practices in software development, emphasizing testing for quality assurance and containerization for enhanced deployment flexibility.

LEARNING OUTCOMES

1. MVC Architecture Mastery: Gain a comprehensive understanding of the Model-View-Controller (MVC) architectural pattern and its application in structuring and organizing web applications.
2. Middleware Expertise: Acquire skills in implementing middleware functions, both global and specific, for tasks such as authentication checks, request parsing, and dynamic view rendering.
3. JWT Token Authentication Proficiency:Develop expertise in implementing JSON Web Tokens (JWT) for secure user authentication, including token creation, verification, and management for maintaining user sessions.
4. Routing and Controller Logic: Master the concepts of routing and controller logic in web development, including defining routes, handling HTTP requests, and structuring application flow.
5. MongoDB Interaction Skills: Learn to interact with MongoDB, a NoSQL database, using Mongoose for data modeling, schema creation, CRUD operations, and establishing robust database connections.
6. Express.js Framework Utilization: Understand the fundamentals of configuring a Node.js server using the Express.js framework, including middleware setup, route definition, and handling HTTP requests.

7. Dynamic View Rendering with EJS: Develop proficiency in using Embedded JavaScript (EJS) as a template engine for dynamically rendering HTML pages based on controller actions, enabling the creation of dynamic and reusable views.
8. Effective Configuration Management: Acquire skills in managing configurations by separating sensitive information, such as database credentials, into a dedicated config.env file, ensuring security and flexibility.
9. RESTful API Design Principles: Learn the principles of designing RESTful APIs, including adherence to HTTP methods, URI patterns, and maintaining statelessness for efficient client-server communication.
10. Custom Middleware Implementation: Master the implementation of custom middleware functions, exemplified by authenticate and authorize, extending the functionality of an Express.js application for user authentication and authorization.

CONSTRUCTIVE FEEDBACK

1. Implement Logging: Integrate logging mechanisms throughout the application to track and monitor critical events. This can aid in debugging, performance optimization, and providing valuable insights into the application's behavior.
2. Optimize DB Queries : Review and optimize database queries, especially in the controllers, to minimize response times. Consider indexing and caching strategies to improve overall database performance.
3. Integrate Front-End Frameworks: Explore the possibility of integrating front-end frameworks like React or Vue.js to create a more interactive and dynamic user interface.

This can improve user experience and simplify the management of complex UI components.

4. Consider API Versioning: Implement API versioning to ensure backward compatibility while making updates. This allows for a smoother transition when introducing new features or modifying existing endpoints.

REFERENCES

Devfolio. (n.d.). *Book Exchange | Devfolio*. https://devfolio.co/projects/book-exchange-4be5

GeeksforGeeks. (2022, June 2). *Model View Controller MVC architecture for Node applications*. https://www.geeksforgeeks.org/model-view-controllermvc-architecture-for-node-applications/

Sufiyan, T. (2023, May 16). *What is Node.js: A Comprehensive Guide*. Simplilearn.com. https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs

MongoDB. (n.d.). *Why use MongoDB and when to use it?* https://www.mongodb.com/why-use-mongodb

How to build middleware for Node.js: A complete guide. (2022, May 28). https://www.turing.com/kb/building-middleware-for-node-js

Build and run a Node.js app in a container. (2021, November 3). https://code.visualstudio.com/docs/containers/quickstart-node

Waeosri, W. (2022, March 3). How to test REST API with Visual Studio Code — REST Client extensions. *Medium*. https://medium.com/lseg-developer-community/how-to-test-rest-api-with-visual-studio-code-rest-client-extensions-9f2e061d0299

Svirca, Z. (2023, September 7). Everything you need to know about MVC architecture - Towards Data Science. *Medium*. https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1

Olusola, S. (2021, December 23). *How to use EJS to template your Node.js application - LogRocket Blog*. LogRocket Blog. https://blog.logrocket.com/how-to-use-ejs-template-node-js-application/

APPENDICES

APPENDIX A

REST API DESCRIPTIONS

Endpoint	HTTP Method	Request	Response
/user	POST	{name:xxxx, Email:xxxx, Password:xxxx, confirmPassword:xxxx }	{ "success": true, "user": { /* user details */ } }
/login	POST	{ Email:xxxx, Password:xxxx}	{ "success": true, "user": { /* user details */ } }
/books	GET		{ "success": true, "books": [/* list of books */]}
/books:id	GET	{id:xx }	{ "success": true, "books": {details of book with id} }
/books	POST	{ "title": "Book Title", "author": "Author Name", "genre":	{ "success": true, "book": { /* book details */ } }

		"Genre", "description": "Description", "owner": "user_id", "exchangeType": "sell", "price": 10 }	
/books:id	PUT	{ "title": "Updated Title", "author": "Updated Author", "description": "Updated Description", "exchangeType": "exchange", "price": 5 }	{ "success": true, "book": { /* updated book details */ }
/books:id	DELETE		{ "success": true, "message": "Book deleted successfully" }

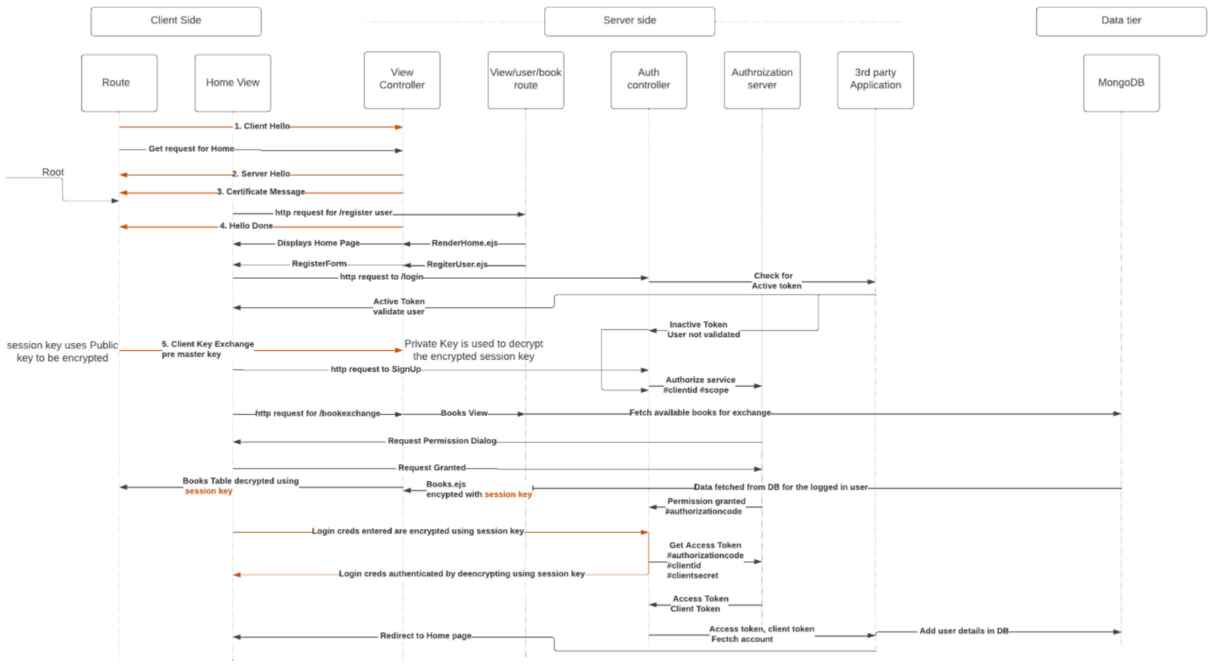
APPENDIX B

UI COMPONENTS

- Navigation Bar: Provides different options – Login, Register, Book Exchange
    - Register – Allows new users to register. A registration page is popped up, which gives a form for entering data.
    - Login – Allows registered users to enter their email address and password and get validated.
    - Book Exchange – A form that will give the users an option to register their books that are up for sale.
- Book listing : Numbers the number of available books and lists the available books with the book information and user information of the user who listed that book.
- Book Details : When each is clicked, a splash screen appears which shows its information and user information.
- User Profile components: View user information – photo, name, email id, username, book lists.

APPENDIX C

SEQUENCE DIAGRAM WITH mTLS AND OAuth



Client Side
Server side
Data tier
Route
Home View
View Controller
View/user/book route
Auth controller
Authroization server
3rd party Application
MongoDB
1. Client Hello
Get request for Home
Root
2. Server Hello
3. Certificate Message
http request for /register user
4. Hello Done
Displays Home Page
RenderHome.ejs
RegisterForm
RegiterUser.ejs
http request to /login
Check for Active token
Active Token validate user
Inactive Token User not validated
session key uses Public key to be encrypted
5. Client Key Exchange pre master key
Private Key is used to decrypt the encrypted session key
http request to SignUp
Authorize service #clientid #scope
http request for /bookexchange
Books View
Fetch available books for exchange
Request Permission Dialog
Request Granted
Books Table decrypted using session key
Books.ejs encypted with session key
Data fetched from DB for the logged in user
Permission granted #authorizationcode
Login creds entered are encrypted using session key
Get Access Token #authorizationcode #clientid #clientsecret
Login creds authenticated by deencrypting using session key
Access Token Client Token
Access token, client token Fectch account
Add user details in DB
Redirect to Home page