FINAL PROJECT

Submited by,

Name: T.Nishalini

Reg.no:922323104026

PH.NO:7448309753

I. General User Tasks (Public Endpoints)

These endpoints are generally accessible to all users without authentication.

Task No.	Description	API Endpoint (Example)	Required Method
Task 1	Get the book list available in the shop.	`/books`	`GET`
Task 2	Get the books based on ISBN.	`/books/isbn/:isbn`	`GET`
Task 11	Search by ISBN – Using Promises.	`/books/isbn/:isbn`	`GET` (Implemented with Promises)
Task 12	Search by Author.	`/books/author/:author`	`GET`
Task 13	Search by Title.	`/books/title/:title`	`GET`
Task 3	Get all books by Author.	`/books/author/:author`	`GET`
Task 4	Get all books based on Title.	`/books/title/:title`	`GET`
Task 5	Get book Review (for a specific book).	`/review/:isbn`	`GET`

II. Authentication Tasks (User Management)

Task No.	Description	API Endpoint (Example)	Required Method
Task 6	Register New User.	`/register`	`POST`
Task 7	Login as a Registered User.	`/login`	`POST`

III. Registered User Tasks (Protected Endpoints)

Task No.	Description	API Endpoint (Example)	Required Method
Task 8	Add/Modify a book review.	`/review/:isbn`	`PUT` (for Add or Modify)
Task 9	Delete book review added by that particular user.	`/review/:isbn`	`DELETE`

IV. Node.js Programming Requirements

Task No.	Description	Technical Requirement
Task 9	Node.JS program with 4 methods (for Add/Modify/Delete review).	Implement the endpoints using Express and a database/data store.
Task 9	Use Async/Await or Promises with Axios in Node.js for all four methods.	All data operations (GET, POST, PUT, DELETE) must use `async/await` syntax with `axios` for API calls.
Task 10	Get all Books – Using async callback function.	Implement a data fetching function for `/books` using a traditional Node.js callback pattern (e.g., `(err, data) => {}`).
Task 11	Search by ISBN – Using Promises.	Implement the data fetching for this search using the Promise pattern (e.g., `.then().catch()`).

Task 11: Search by ISBN Using Promises

Requirement: Implement a function in Node.js that fetches book data by ISBN using the Promise pattern (`.then()` and `.catch()`).

Assumed Endpoint: `http://localhost:5000/books/isbn/`

```
// Import the necessary library

const axios = require('axios');



// Define the base URL of your API

const BASE_URL = 'http://localhost:5000';



/**

 * Fetches a book's details using its ISBN, implemented with Promises.

 * @param {string} isbn The International Standard Book Number to search for.

 * @returns {Promise<object>} A Promise that resolves with the book data or rejects with an error.

 */

function getBookByISBN(isbn) {

  const url = `${BASE_URL}/books/isbn/${isbn}`;

  console.log(`Searching for book at: ${url}`);



  return new Promise((resolve, reject) => {

    // 1. Make the API call using axios

    axios.get(url)

      .then(response => {

        // 2. Resolve the promise with the successful data

        if (response.data) {

          resolve(response.data);
        } else {
          // Handle cases where the API returns 200 but no data
          reject(new Error(`Book with ISBN ${isbn} not found.`));
        }
      })
      .catch(error => {
        // 3. Reject the promise on network errors or non-200 status codes
        // Check if the error has a response object for better debugging
        if (error.response) {
          reject(new Error(`API Error: ${error.response.status} - 
${error.response.data.message || 'Server responded with error.'}`));
        } else if (error.request) {
          reject(new Error('Network Error: No response received from server.'));
        } else {
          reject(new Error(`Request Setup Error: ${error.message}`));
        }
      });
  });
}


// --- Example Usage (Demonstrating the Promise Pattern) ---


const sampleISBN = '978-0743273565';


getBookByISBN(sampleISBN)
  .then(bookData => {
    console.log('\n✅ Success! Book Found:');
    console.log(`Title: ${bookData.title}`);
    console.log(`Author: ${bookData.author}`);
  })
  .catch(error => {
    console.error('\n❌ Error during search:');
    console.error(error.message);
  });
```

Task 10: Get All Books Using Async Callback

Requirement: Create a function that fetches all books and uses a standard callback function `(error, data)` to return the result.

Assumed Endpoint: `http://localhost:5000/books`

```
// Import the necessary library (We'll still use Axios, but wrap it to use a callback)
const axios = require('axios');


// Define the base URL of your API
const BASE_URL = 'http://localhost:5000';


/**
 * Fetches the entire book list using an asynchronous callback function.
 * @param {function(Error|null, Array<object>|null): void} callback
 * The callback function to handle the result.
 * It takes (error, data) as arguments.
 */
function getAllBooks(callback) {
  const url = `${BASE_URL}/books`;

  console.log(`Fetching all books from: ${url}`);



  // 1. Make the API call

  axios.get(url)

    .then(response => {

      // 2. On success, call the callback with no error (null) and the data

      console.log('Successfully received data.');

      callback(null, response.data);

    })

    .catch(error => {

      // 3. On failure, call the callback with the error and no data (null)

      let errorMessage;

      if (error.response) {

        errorMessage = `API Error: ${error.response.status} - ${error.response.data.message
|| 'Server error.'}`;

      } else {

        errorMessage = `Network or Request Error: ${error.message}`;

      }

      console.error('Error during data fetch.');

      callback(new Error(errorMessage), null);

    });

}



// --- Example Usage (Demonstrating the Callback Pattern) ---



getAllBooks((err, bookList) => {
  if (err) {
    // Handle the error case
    console.error('\n❌ Failed to retrieve books:');
    console.error(err.message);
    return;
  }


  // Handle the success case
  console.log('\n✅ Successfully retrieved book list!');
  console.log(`Total books found: ${bookList.length}`);


  // Optionally display the first book's title
  if (bookList.length > 0) {
      console.log(`First book title: ${bookList[0].title}`);
  }
});
```

Task 9: Add/Modify Review Using Async/Await

Requirement: Implement the review update/creation logic using `async/await`. This assumes the user is authenticated (e.g., a JWT is included in the request headers).

Assumed Endpoint: `http://localhost:5000/review/:isbn`

Method: `PUT` **(Used for both adding a new review or modifying an existing one for the same user).**

```
// Import the necessary library
const axios = require('axios');


// Define the base URL and a placeholder token for authentication
const BASE_URL = 'http://localhost:5000';
const USER_TOKEN = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'; // Replace with a
real token


/**
 * Adds or modifies a book review for a specific user and ISBN using async/await.
 * @param {string} isbn The ISBN of the book.
 * @param {string} reviewText The content of the review.
 * @returns {Promise<object>} The response data from the server.
 */
async function addOrModifyReview(isbn, reviewText) {
  const url = `${BASE_URL}/review/${isbn}`;

  // 1. Define the data payload to send in the PUT request
  const reviewData = {
    review: reviewText
  };

  // 2. Define the configuration, including the required authentication header
  const config = {
    headers: {
      'Authorization': `Bearer ${USER_TOKEN}`, // Used to identify the registered user
      'Content-Type': 'application/json'
    }
  };

  try {
    console.log(`Sending PUT request to: ${url}`);
    // 3. Use 'await' to pause execution until the promise resolves
    const response = await axios.put(url, reviewData, config);


    // 4. Return the data on success
    console.log('✅ Review successfully added/modified.');
    return response.data;


  } catch (error) {
    // 5. Handle errors thrown by axios (e.g., 401 Unauthorized, 404 Not Found)
    if (error.response) {
      console.error(`❌ API Error: ${error.response.status} - ${error.response.data.message || 'Server error.'}`);
      throw new Error(error.response.data.message || `Failed to update review for ISBN ${isbn}.`);
    } else {
      console.error(`❌ Network Error: ${error.message}`);
      throw new Error(`Could not connect to the API server.`);
    }
  }
}


// --- Example Usage (Using the async function) ---


// Self-invoking function to run the async logic
(async () => {
  const targetISBN = '978-0385537858';
  const newReview = "An absolutely captivating read, highly recommended!";


  try {
    const result = await addOrModifyReview(targetISBN, newReview);
    console.log('\nFinal API Response:', result);
  } catch (e) {
    console.error(`\nOperation failed: ${e.message}`);
  }
})();
```

Task 9: Delete Book Review Using Async/Await

Requirement: Implement the logic to delete a user's own review for a specific book using `async/await`. This task assumes the server identifies the user via the authentication token and only allows them to delete their own review.

Assumed Endpoint: `http://localhost:5000/review/:isbn`

Method: `DELETE`

```
// Import the necessary library
const axios = require('axios');


// Define the base URL and a placeholder token for authentication
const BASE_URL = 'http://localhost:5000';
const USER_TOKEN = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'; // Replace with a real token


/**
 * Deletes a book review for a specific user and ISBN using async/await.
 * @param {string} isbn The ISBN of the book whose review should be deleted.
 * @returns {Promise<object>} The success message from the server.
 */
async function deleteReview(isbn) {
  const url = `${BASE_URL}/review/${isbn}`;

  // 1. Define the configuration, including the required authentication header
  const config = {
    headers: {
      'Authorization': `Bearer ${USER_TOKEN}`, // Used to identify the registered user
      'Content-Type': 'application/json'
    }
  };

  try {
    console.log(`Sending DELETE request to: ${url}`);

    // 2. Use 'await' to pause execution until the DELETE promise resolves
    const response = await axios.delete(url, config);

    // 3. Return the data on success (usually a confirmation message)
    console.log(`✅ Review for ISBN ${isbn} successfully deleted.`);
    return response.data;

  } catch (error) {
    // 4. Handle errors (e.g., 401 Unauthorized, 404 Review Not Found)
    if (error.response) {
      const status = error.response.status;
      const message = error.response.data.message || 'Server error occurred.';
      console.error(`❌ API Error (${status}): ${message}`);
      throw new Error(message);
    } else {
      console.error(`❌ Network Error: ${error.message}`);
      throw new Error(`Could not connect to the API server.`);
    }
  }
}

// --- Example Usage (Running the async function) ---

// Self-invoking function to run the async logic
(async () => {
  const targetISBN = '978-0385537858'; // The ISBN of the book

  try {
    const result = await deleteReview(targetISBN);
    console.log('\nFinal Deletion API Response:', result);
  } catch (e) {
    console.error(`\nDeletion failed: ${e.message}`);
  }
})();
```

Task 6: Register New User

This task requires sending a user's chosen username and password to the server to create a new account.

Endpoint: `http://localhost:5000/register` **Method:** `POST`

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

const BASE_URL = 'http://localhost:5000';



/**

 * Registers a new user account with the provided credentials using async/await.

 * @param {string} username The desired username.

 * @param {string} password The user's chosen password.

 * @returns {Promise<object>} The server's registration success message.

 */

async function registerUser(username, password) {

  const url = `${BASE_URL}/register`;



  const userData = {

    username: username,

    password: password

  };



  try {

    console.log(`Attempting to register user: ${username}`);



    // POST request sends the user data in the body

    const response = await axios.post(url, userData);
    console.log('✅ User registered successfully.');
    return response.data;

  } catch (error) {
    if (error.response) {
      // Handle status codes like 409 Conflict (User already exists) or 400 Bad Request
      const status = error.response.status;
      const message = error.response.data.message || 'Registration failed.';
      console.error(`❌ Registration Error (${status}): ${message}`);
      throw new Error(message);
    } else {
      throw new Error(`Could not connect to the API server during registration.`);
    }
  }
}

// --- Example Usage ---
(async () => {
  try {
    const result = await registerUser('testuser123', 'MySecurePass!');
    console.log('\nRegistration Result:', result);
  } catch (e) {
    console.error(`\nOperation failed: ${e.message}`);
  }
})();
```

Task 7: Login as a Registered User

This task requires sending the user's username and password to the server to verify credentials and receive an authentication token (like a JWT).

Endpoint: `http://localhost:5000/login` **Method:** `POST`

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

const BASE_URL = 'http://localhost:5000';



/**

 * Logs in a registered user and retrieves the authentication token.

 * @param {string} username The user's username.

 * @param {string} password The user's password.

 * @returns {Promise<string>} The authentication token received from the server.

 */

async function loginUser(username, password) {

  const url = `${BASE_URL}/login`;



  const credentials = {

    username: username,

    password: password

  };



  try {

    console.log(`Attempting to log in user: ${username}`);



    const response = await axios.post(url, credentials);



    // The token is usually nested in the response data
    const token = response.data.token;


    if (token) {
        console.log('✅ Login successful. Token retrieved.');
        return token;
    } else {
        // Handle successful request but missing token (bad server response)
        throw new Error("Login failed: Server did not return a token.");
    }


  } catch (error) {
    if (error.response) {
      // Handle 401 Unauthorized (invalid credentials)
      const status = error.response.status;
      const message = error.response.data.message || 'Login failed.';
      console.error(`❌ Login Error (${status}): ${message}`);
      throw new Error(message);
    } else {
      throw new Error(`Could not connect to the API server during login.`);
    }
  }
}


// --- Example Usage ---
(async () => {
  try {
    const receivedToken = await loginUser('testuser123', 'MySecurePass!');

    console.log('\nAuthentication Token (JWT):', receivedToken);



    // This token would then be used in subsequent requests (Tasks 8 & 9)

    // to authenticate the user.



  } catch (e) {

    console.error(`\nOperation failed: ${e.message}`);

  }

})();
```

14: Submission of Project GitHub Link

This task doesn't require any code, but it is the crucial final step to submit your work for grading.

📝Key Requirements for Your GitHub Repository

Before submitting the link, ensure your GitHub repository contains all the necessary components for your peer reviewer to grade your work:

Source Code: All the Node.js files (server code, authentication logic, API endpoints, etc.) used to fulfill Tasks 1 through 13.

Screenshots Folder: A folder containing all the required screenshots from the lab environment, including the ones showing successful execution of Tasks 1, 3, and any others specified in the course (e.g., Postman/API client results).

`README.md` **File: This is the most critical document. It should clearly explain:**

Project Title and Description.

Installation Instructions: How to clone the repo, install dependencies (`npm install`), and start the server (`npm start` or similar).

API Documentation: A brief list or table (similar to the one provided earlier) detailing the API endpoints (`/books`, `/login`, `/review/:isbn`) and the HTTP methods (`GET`, `POST`, `PUT`, `DELETE`) used for each task. This helps the reviewer verify your work.

✅ **Submission Process**

The "document" you submit for Task 14 will be the direct URL to your public GitHub repository:

Commit and Push: Ensure all your latest changes, code, and documentation are committed and pushed to the main branch of your GitHub repository.

Verify Access: Make sure the repository is Public so the reviewer can access it without needing login credentials.

Copy the URL: Copy the URL (e.g., `https://github.com/YourUsername/YourProjectName`).

Paste and Submit: Enter this URL into the submission box for Task 14 in the lab environment.