Book Review API

Overview

This is a RESTful API built with Node.js and Express.js for a basic Book Review system. It includes user authentication, book management, and review functionalities.

Tech Stack

- Node.js
- Express.js
- MongoDB (DB)
- JWT (JSON Web Tokens) for Authentication

Features

- **Authentication:**
  - User registration and login using JWT.
- **Book Management:**
  - Add new books (protected).
  - Retrieve all books with pagination and filtering.
  - Retrieve a single book with details, average rating, and paginated reviews.
- **Review Management:**
  - Submit reviews for books (protected, one review per user per book).
  - Update/delete own reviews.
- **Search**
  - Search books by title or author

Database Schema

MongoDB Schema

1. User Model

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



const userSchema = new mongoose.Schema({

  username: { type: String, required: true },

  email: { type: String, unique: true, required: true },
  password: { type: String, required: true }

});



module.exports = mongoose.model('User', userSchema);
```

2. Book Model

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



const bookSchema = new mongoose.Schema({

  title: String,

  author: String,

  genre: String,

  reviews: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Review' }]

});



module.exports = mongoose.model('Book', bookSchema);
```

3. Review Model

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



const reviewSchema = new mongoose.Schema({

  book: { type: mongoose.Schema.Types.ObjectId, ref: 'Book', required:
```

```
true },

  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },

  rating: { type: Number, required: true },

  comment: String

});


module.exports = mongoose.model('Review', reviewSchema);
```

Project Setup

1. **Clone the repository:**

2. **Install dependencies:**
   npm install

3. **Set up environment variables:**
   - In the .env file i created i just made MONGO_URI different change to your connection string after creating connection string from atlas
4. **Run the application:**
   Node server.js

   The server will start at http://localhost:3000 (or the port specified in your .env file).

How to Run Locally

See "Project Setup Instructions" above.

Example API Requests

USE postman if not installed install

1. User Authentication

- Postman:
    - Method: POST
    - URL: http://localhost:3000/signup
    - Headers: Content-Type: application/json
    - Body:
      ```
      {
         "username": "kozhi",
         "email": "kozhi@gmail.com",
         "password": "password123"
      }
      ```

- Postman:
    - Method: POST
    - URL: http://localhost:3000/login
    - Headers: Content-Type: application/json
    - Body:
      ```
      {
         "email": "kozhi@gmail.com",
         "password": "password123"
      }
      ```

    - Response:
      ```
      {
         "token": "jwt_token"  // Copy token }
      ```

2. Book Management

POST /books

- Postman:
    - Method: POST
    - URL: http://localhost:3000/books
    - Headers:
        - Content-Type: application/json
        - Authorization: Bearer <token> (Replace <token>)
    - Body:
      ```
      {
         "title": "The Great Gatsby",
         "author": "F. Scott Fitzgerald",
         "genre": "Classic",
      ```

```
    "description": "A novel about wealth, love, and the American Dream."
}
```

GET /books

- Curl:
  curl http://localhost:3000/books?page=1&limit=10&author=Scott

- Postman
  - Method: GET
  - URL: http://localhost:3000/books?page=1&limit=10&author=Scott
  - (Add query parameters in the Params tab)
    - page: 1
    - limit: 10
    - author: Scott

GET /books/:id

- Postman:
  - Method: GET
  - URL: http://localhost:3000/books/65e571e49b8b4b7d1c5e57a8 (Replace with a valid book ID)

GET /search

- Postman
  - Method: GET
  - URL: http://localhost:3000/search?query=Gatsby

3. Review Management

POST /books/:id/reviews

- Postman:
  - Method: POST
  - URL: http://localhost:3000/books/65e571e49b8b4b7d1c5e57a8/reviews (Replace with a valid book ID)
  - Headers:
    - Content-Type: application/json
    - Authorization: Bearer <your_jwt_token>
  - Body:
    ```
    {
      "rating": 5,
    ```

```
  "comment": "A fantastic book!"
}
```

PUT /reviews/:id

- Postman
    - Method: PUT
    - URL: http://localhost:3000/reviews/65e589f39b8b4b7d1c5e57b1
    - Headers:
        - Content-Type: application/json
        - Authorization: Bearer <your_jwt_token>
    - Body:
      ```
      {
        "rating": 4,
        "comment": "It was good"
      }
      ```

DELETE /reviews/:id

- Postman:
    - Method: DELETE
    - URL: http://localhost:3000/reviews/65e589f39b8b4b7d1c5e57b1 (Replace with a valid review ID)
    - Headers:
        - Authorization: Bearer <your_jwt_token>

Design Decisions and Assumptions

- **Database:** MongoDB was chosen for its flexibility in handling JSON-like data, which is common in web APIs.
- **Authentication:** JWT was chosen for its stateless nature, scalability, and ease of implementation.
- **Error Handling:** Basic error handling is included (e.g., 400 for bad requests, 401 for authentication errors, 500 for server errors). More robust error handling could be implemented.
- **Validation:** Basic input validation is implemented. Consider using a library like Joi for more comprehensive validation.
- **Pagination:** Pagination is implemented for the /books and /books/:id/reviews endpoints to prevent overwhelming the client with large datasets. The defaults are set in the controller.

- **Reviews:**
  - Users can only submit one review per book. This is enforced by a unique index on the user and book fields in the Review model.
- **Security:**
  - Password hashing is used (with bcrypt).
  - The JWT_SECRET should be stored securely in an environment variable.
  - Input validation and sanitization should be used to prevent injection attacks.
- **Search:**
  - The search functionality in the GET /search endpoint uses a case-insensitive, partial string search on the book title and author.
- **Assumptions:**
  - The application assumes a basic understanding of RESTful API principles.
  - Date is stored as a Date object in Mongodb.