Documentation

Authentication and Role-Based Access Control (RBAC) Documentation

This document explains how authentication and role-based access control (RBAC) are implemented and how they can be tested in the Books Collection API and more.

Authentication

The API uses **JWT (JSON Web Tokens)** for user authentication. Users must sign up or log in to obtain a token. This token is required to access protected routes.

Authentication Process

1. **Sign Up**
    - ✓ **Endpoint**: POST /auth/signup
    - ✓ **Purpose**: Create a new user account.
    - ✓ **Request Body**:

```
{
  "username": "exampleUser",
  "password": "securePassword",
  "role": "admin" // Optional, defaults to "user"
}
```

    - ✓ **Response**:

```
{
  "message": "User created successfully"
}
```

2. **Log In**
    - ✓ **Endpoint**: POST /auth/login
    - ✓ **Purpose**: Authenticate a user and retrieve a JWT token.

- ✓ **Request Body**:

```
{
  "username": "exampleUser",
  "password": "securePassword"
}
```

- ✓ **Response**:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." as an example
}
```

3. **Token Usage**
    - ✓ Include the token in the Authorization header for all protected routes:

Authorization: Bearer <token>

Role-Based Access Control (RBAC)

RBAC ensures that users can only access resources allowed for their roles. This API supports two roles:

- **Admin**
- **User**

RBAC Implementation

1. **Roles Assignment**
    - Roles are assigned during user creation.
    - The default role is user.
    - Example of an admin user:

```
{
  "username": "adminUser",
```

"password": "securePassword",

"role": "admin"

}

2. **Middleware**

    - **Authentication Middleware**: Ensures the user is authenticated before accessing protected routes.
    - **Role Middleware**: Checks if the authenticated user has the required role to access a route.

Protected Routes

Route	Method	Role	Description
/books/all	GET	Admin	Fetch all books.
/books	GET	User/Admin	Fetch books accessible to the logged-in user.
/books	POST	User/Admin	Add a new book.
/books/:id	PUT	User/Admin	Update a book by ID.
/books/:id	DELETE	Admin	Delete a book by ID.

Testing Authentication and RBAC

Using Postman

1. **Sign Up**:

    - Send a POST request to /auth/signup with a username, password, and role.
    - Example:

{

"username": "testUser",

"password": "123456",

"role": "admin"

}

2. **Log In**:
    - Send a POST request to /auth/login with the same username and password to retrieve a JWT token.
3. **Access Protected Routes**:
    - Use the token in the Authorization header to test protected routes.
    - Example Header:

Authorization: Bearer <token>

4. **Role-Specific Routes**:
    - Test admin-only routes like /books/all with an admin token.
    - Try accessing it with a user token to verify the role restrictions.

Error Responses

- **Unauthenticated User**:

```
{
  "error": "Access Denied"
}
```

- **Unauthorized Role**:

```
{
  "error": "You do not have the required permissions to access this route"
}
```

- **Token Expired/Invalid**:

```
{
  "error": "Invalid or expired token"
}
```

Security Features

1. **Password Hashing**:
    - Passwords are hashed using bcrypt before being stored in the database.
2. **JWT Secret**:
    - The token is signed using a secret stored in the environment variable JWT_SECRET.
3. **Route Protection**:
    - Routes are protected by middleware to ensure only authenticated users with the correct roles can access them.

Books Collection API Documentation

Base URL

For local development:

http://localhost:5000/books

For deployment:

https://stage-3-books-collection-api-updated-7.onrender.com/auth/signup

Books Endpoints

1. Get All Books (Admin Only)

- **Endpoint**: GET /books/all
- **Description**: Fetches all books.
- **Authorization**: Requires admin role.

Headers:

Authorization: Bearer <jwt-token>

Response:


```
[
 {
  "_id": "bookId1",
```

```
  "title": "Book Title",
  "author": "Author Name",
  "isbn": "123456789",
  "publishedYear": 2021
 },
 {
  "_id": "bookId2",
  "title": "Another Book Title",
  "author": "Another Author Name",
  "isbn": "987654321",
  "publishedYear": 2020
 }
]
```


2. Get Books (User/Admin)

- **Endpoint**: GET /books
- **Description**: Fetches books accessible to the logged-in user.

Headers:

Authorization: Bearer <jwt-token>

Response:


```
[
 {
  "_id": "bookId1",
  "title": "Book Title",
  "author": "Author Name",
```

```
    "isbn": "123456789",
    "publishedYear": 2021
  }
]
```


3. Add a Book

- **Endpoint**: POST /books
- **Description**: Adds a new book.
- **Authorization**: Requires a valid user or admin token.

Headers:

Authorization: Bearer <jwt-token>

Request Body:


```
{
  "title": "New Book",
  "author": "Author Name",
  "isbn": "123456789",
  "publishedYear": 2022
}
```


Response:


```
{
  "message": "Book added successfully",
  "book": {
    "_id": "bookId",
    "title": "New Book",
    "author": "Author Name",
```

```
    "isbn": "123456789",
    "publishedYear": 2022
  }
}
```


4. Get a Book by ID

- **Endpoint**: GET /books/:id
- **Description**: Fetches a book by its ID.

Response:


```
{
  "_id": "bookId",
  "title": "Book Title",
  "author": "Author Name",
  "isbn": "123456789",
  "publishedYear": 2021
}
```


5. Update a Book by ID

- **Endpoint**: PUT /books/:id
- **Description**: Updates a book by its ID.
- **Authorization**: Requires a valid user or admin token.

Headers:

Authorization: Bearer <jwt-token>

Request Body:


```
{
  "title": "Updated Title",
```

```
  "author": "Updated Author",
  "isbn": "123456789",
  "publishedYear": 2023
}
```


Response:


```
{
  "message": "Book updated successfully",
  "book": {
    "_id": "bookId",
    "title": "Updated Title",
    "author": "Updated Author",
    "isbn": "123456789",
    "publishedYear": 2023
  }
}
```


6. Delete a Book by ID (Admin Only)

- **Endpoint**: DELETE /books/:id
- **Description**: Deletes a book by its ID.
- **Authorization**: Requires admin role.

Headers:

Authorization: Bearer <jwt-token>

Response:


```
{
  "message": "Book deleted successfully"
```

}

Postman Collection

1. Import the provided Postman collection JSON file into Postman.
2. Replace variables like <jwt-token> with actual values from the login response.
3. Test each endpoint using the specified request payloads and headers.

- **Deploy**: I use a Render to deploy.
- https://stage-3-books-collection-api-updated-7.onrender.com/auth/signup