API Technical Documentation

Overview

This document provides the technical specifications for the `myFlix` API, which allows users to manage their movie collections and profiles. The API supports user authentication, user management, and movie management operations.

Table of Contents

Setup

Environment Variables

Ensure to set the following environment variables in your `.env` file:

- `MONGODB_URI`: Connection string for MongoDB.
- `JWT_SECRET`: Secret key for JWT (default is `'a1b2c3d4e5f6'`).
- `PORT`: Port number for the server (default is 8080).

Required Packages

- `express`
- `dotenv`
- `morgan`
- `mongoose`
- `express-validator`
- `body-parser`
- `passport`

- `passport-local`
- `passport-jwt`
- `jsonwebtoken`
- `cors`
- `bcryptjs`

Authentication

Passport Configuration

The authentication mechanism uses Passport.js with two strategies: Local Strategy for username/password authentication and JWT Strategy for token-based authentication.

1. Local Strategy

This strategy authenticates users using a username and password. It validates the user against the database and checks the password using bcrypt.

Example Code:

javascript
Copy code

```
passport.use(new LocalStrategy({
    usernameField: 'username',
    passwordField: 'password',
}, async (username, password, done) => {
    const user = await Users.findOne({ username });
    if (!user || !user.validatePassword(password)) {
        return done(null, false, { message: 'Incorrect username or
password.' });
    }
    return done(null, user);
}));
```

2. JWT Strategy

This strategy verifies the JWT token and retrieves the user from the database.

Example Code:

javascript

Copy code
```
passport.use(new JWTStrategy({
    jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken(),
    secretOrKey: process.env.JWT_SECRET || 'a1b2c3d4e5f6',
}, async (jwtPayload, done) => {
    const user = await Users.findById(jwtPayload._id);
    return done(null, user);
}));
```

Token Generation

A JWT token is generated upon successful login, which can be used for subsequent requests.

Example Code:

javascript
Copy code
```
const generateJWTToken = (user) => {
    return jwt.sign(user, process.env.JWT_SECRET || 'a1b2c3d4e5f6', {
        subject: user.username,
        expiresIn: '7d',
        algorithm: 'HS256',
    });
};
```

Endpoints

User Management

1. Create a User

POST `/users`

Request Body:

json
Copy code
```
{
    "username": "string (required, min 5 chars)",
    "password": "string (required)",
```

```
    "email": "string (valid email, required)",
    "dateOfBirth": "string (format: YYYY-MM-DD, required)"
}
```

Success Response:

- **Code**: `201 Created`

json
Copy code

```
{
    "message": "User created successfully",
    "user": {
        "username": "string",
        "email": "string",
        "dateOfBirth": "string"
    }
}
```

Error Response:

- **Code**: `422 Unprocessable Entity`

json
Copy code

```
{
    "errors": [
        { "msg": "Error message", "param": "field" }
    ]
}
```

2. Update User Information

PUT `/users/:Username`

Request Body (optional):

json
Copy code

```
{
```

```
    "username": "string (optional)",
    "password": "string (optional)",
    "email": "string (optional)",
    "dateOfBirth": "string (optional)"
}
```

Success Response:

- **Code**: `200 OK`

json
Copy code
```
{
    "username": "string",
    "email": "string",
    "dateOfBirth": "string"
}
```

Error Response:

- **Code**: `404 Not Found`

json
Copy code
```
{
    "message": "User not found"
}
```

3. Delete a User

DELETE `/users/:Username`

Success Response:

- **Code**: `200 OK`

json
Copy code
```
{
    "message": "User deleted successfully"
```

```
}
```

Error Response:

- **Code**: `404 Not Found`

json
Copy code
```
{
    "message": "User not found"
}
```

Movie Management

1. Get All Movies

GET `/movies`

Success Response:

- **Code**: `200 OK`

json
Copy code
```
[
    {
        "title": "string",
        "genre": "string",
        "director": "string"
    }
]
```

2. Get Movie by Title

GET `/movies/:title`

Success Response:

- **Code**: `200 OK`

json

Copy code

```
{
    "title": "string",
    "genre": "string",
    "director": "string"
}
```

Error Response:

- **Code**: `404 Not Found`

json
Copy code

```
{
    "message": "Movie not found"
}
```

3. Add Movie to Favorites

PATCH `/users/:Username/movies/:MovieID?`

Request Body:

json
Copy code

```
{
    "MovieID": "string (required)"
}
```

Success Response:

- **Code**: `200 OK`

json
Copy code

```
{
    "favoriteMovies": ["MovieID"]
}
```

4. Delete Movie from Favorites

DELETE `/users/:Username/movies/:MovieID`

Success Response:

- **Code**: `200 OK`

json
Copy code
```
{
    "favoriteMovies": ["MovieID"]
}
```

Error Handling

The API provides standardized error responses for validation and processing errors, including:

- **400 Bad Request**
- **404 Not Found**
- **500 Internal Server Error**

Testing

You can test the API using tools like Postman or cURL.

Postman Setup:

1. Set the request type (GET, POST, PUT, DELETE, PATCH).
2. Enter the appropriate URL.
3. Set headers (e.g., `Content-Type: application/json`).
4. Enter JSON in the body for POST and PUT requests.

cURL Example:

bash
Copy code
```
curl -X POST http://localhost:8080/users \
-H "Content-Type: application/json" \
-d '{
    "username": "testuser",
    "password": "Test@1234",
    "email": "test@example.com",
    "dateOfBirth": "2000-01-01"
```

```
}'
```

Conclusion

This API facilitates the management of user profiles and movies, providing a robust system for a movie collection application. Ensure to follow the guidelines for authentication and error handling to maintain a smooth user experience.