Group 8: All in one OTT Platform
Unit Test

Introduction

- This document presents a comprehensive analysis of the unit testing performed for the project, utilizing advanced tools like Jest, Istanbul,Early AI. It explores back-end testing, providing detailed insights into coverage metrics, testing methodologies, and the benefits these practices bring to software development

Tools and Frameworks

- **Jest**: A robust JavaScript testing framework designed to validate code correctness and ensure reliability across applications.

- **Istanbul**: A powerful tool for generating detailed coverage statistics, helping developers identify untested portions of their JavaScript codebase.

White-Box Testing

White-box testing focuses on analyzing and verifying the internal structure, logic, and mechanics of the codebase. Unlike black-box testing, which examines the external behavior of a system, White-box testing examines the internal operations and structure of the code, enabling developers to validate the integrity of individual code paths and structures.

Importance of Unit Testing

- **Early Bug Detection**: Enables developers to identify and address issues early in the development lifecycle, reducing the potential impact on the project.

- **Improved Code Quality**: Promotes the creation of clean, modular, and maintainable code by encouraging developers to write testable code from the start.

- **Efficient Debugging**: Simplifies the debugging process by providing precise insights into the root cause of failures when tests do not pass.

Backend Unit Testing using Jest

Overall Coverage

File		Statements		Branches		Functions		Lines	
controllers		97.55%	239/245	92.72%	102/110	96.29%	26/27	97.54%	238/244
middlewares		100%	4/4	100%	0/0	100%	3/3	100%	3/3
models		100%	6/6	100%	0/0	100%	0/0	100%	6/6
utils		100%	12/12	100%	2/2	100%	4/4	100%	11/11

Code coverage generated by istanbul at 2024-12-01T11:05:12.286Z

Figure 1: Overall coverage for backend testing.

Coverage of Controllers

All files controllers

97.55% Statements 239/245 **92.72%** Branches 102/110 **96.29%** Functions 26/27 **97.54%** Lines 238/244

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

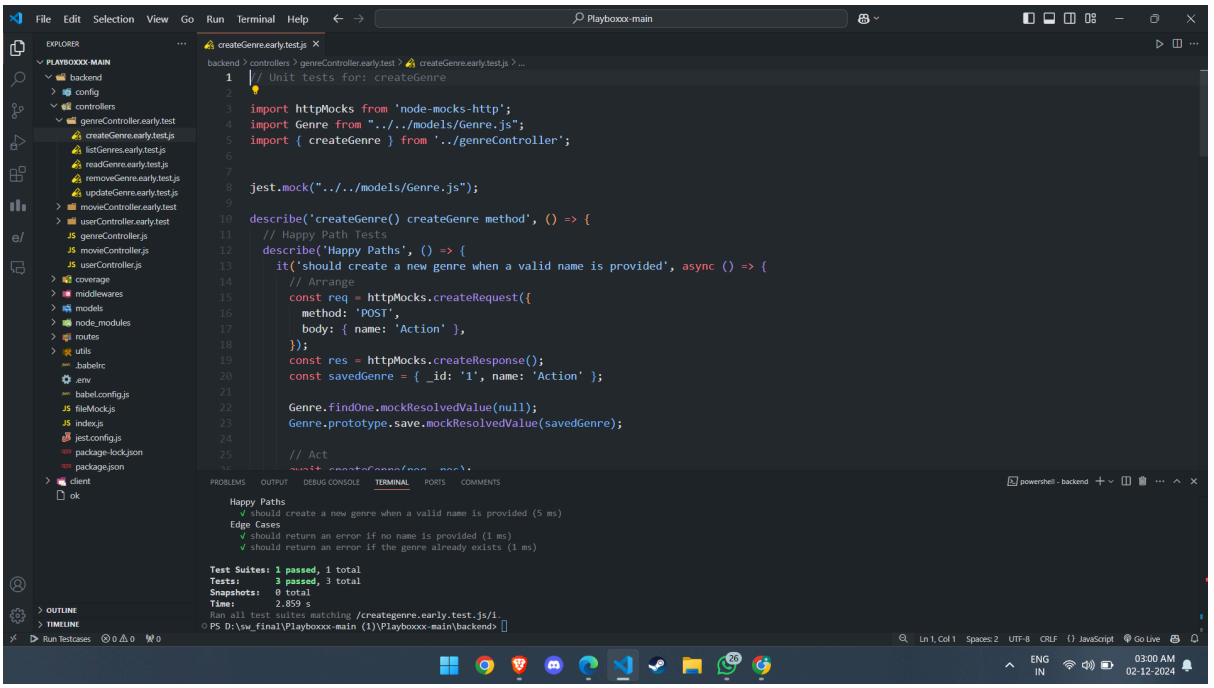
Filter:

File		Statements		Branches		Functions		Lines	
genreController.js		95.55%	43/45	100%	8/8	100%	5/5	95.55%	43/45
movieController.js		98.67%	149/151	90%	72/80	100%	16/16	98.67%	149/151
userController.js		95.91%	47/49	100%	22/22	83.33%	5/6	95.83%	46/48

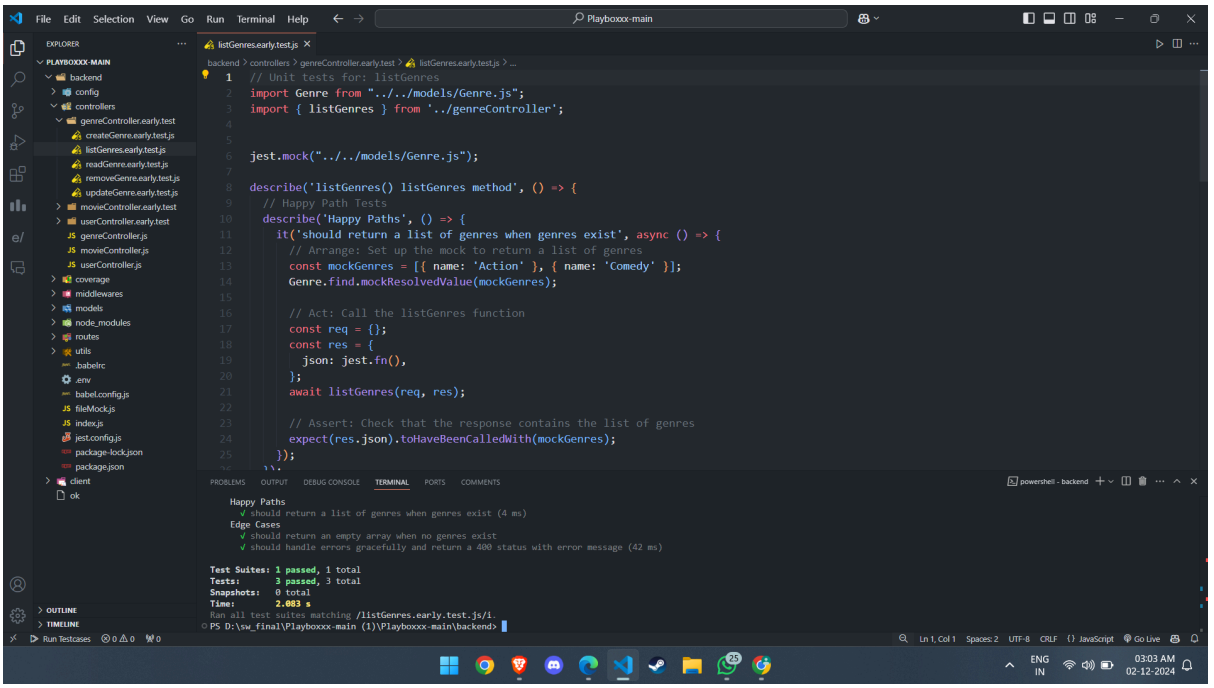
Figure 2: Coverage of backend controllers.

1. Testing of genre Controller

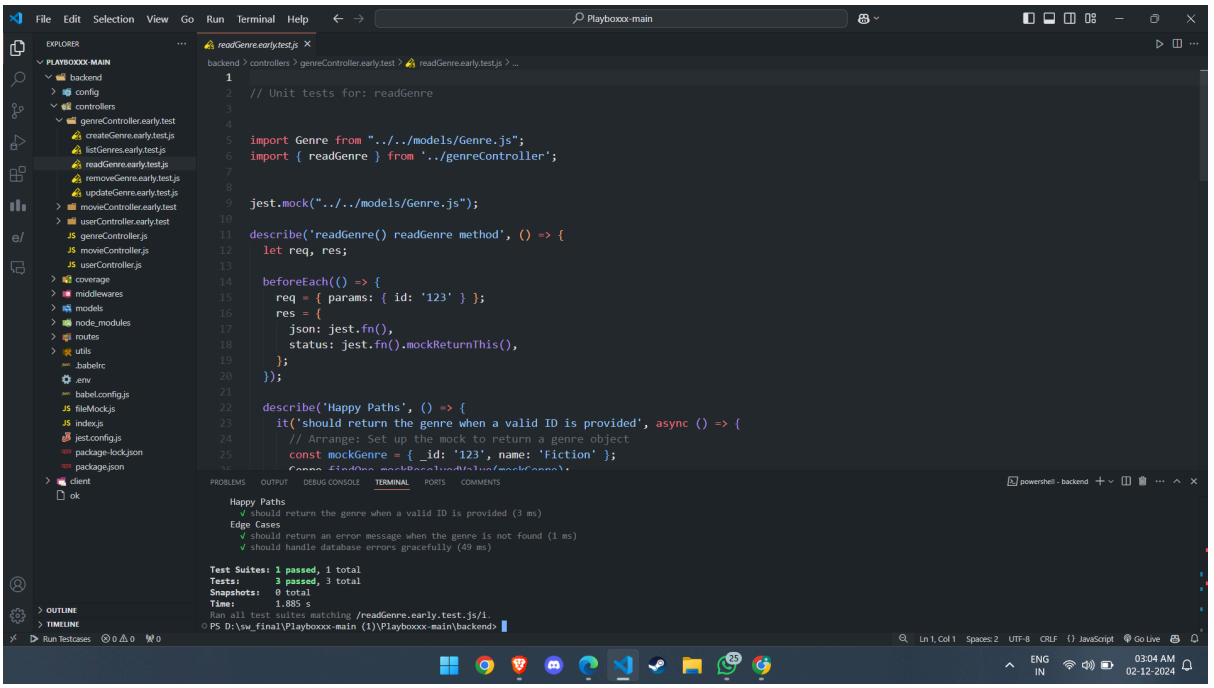
Create genres testing



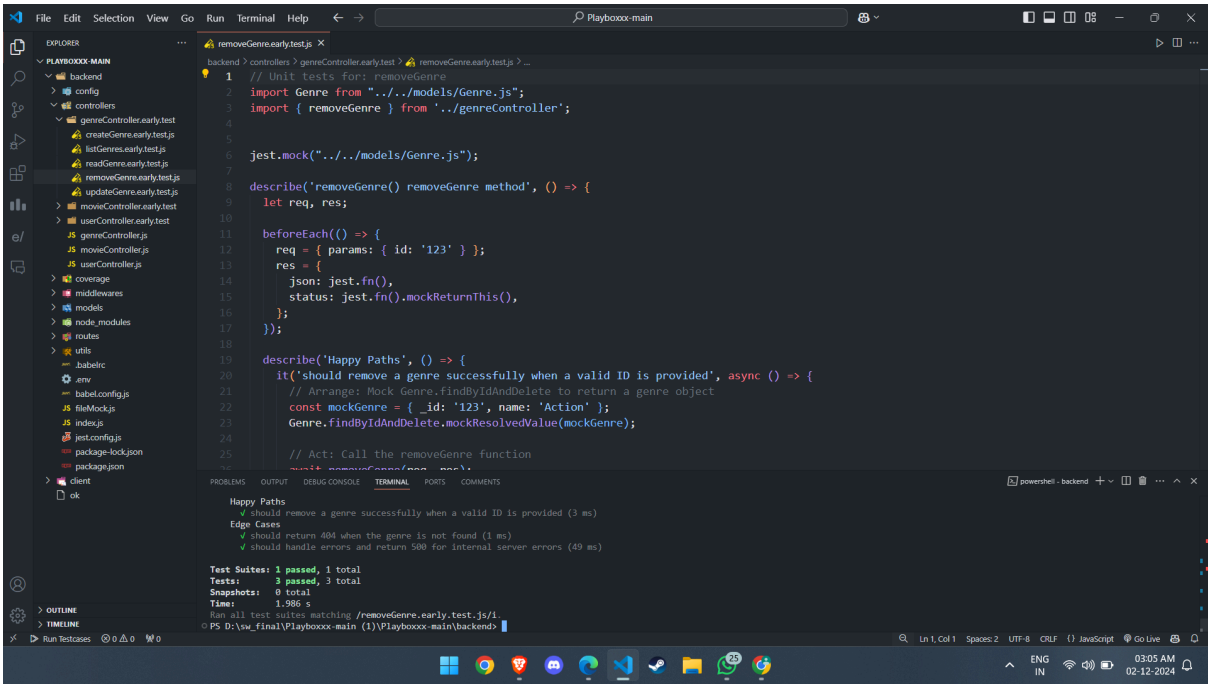
List genres testing



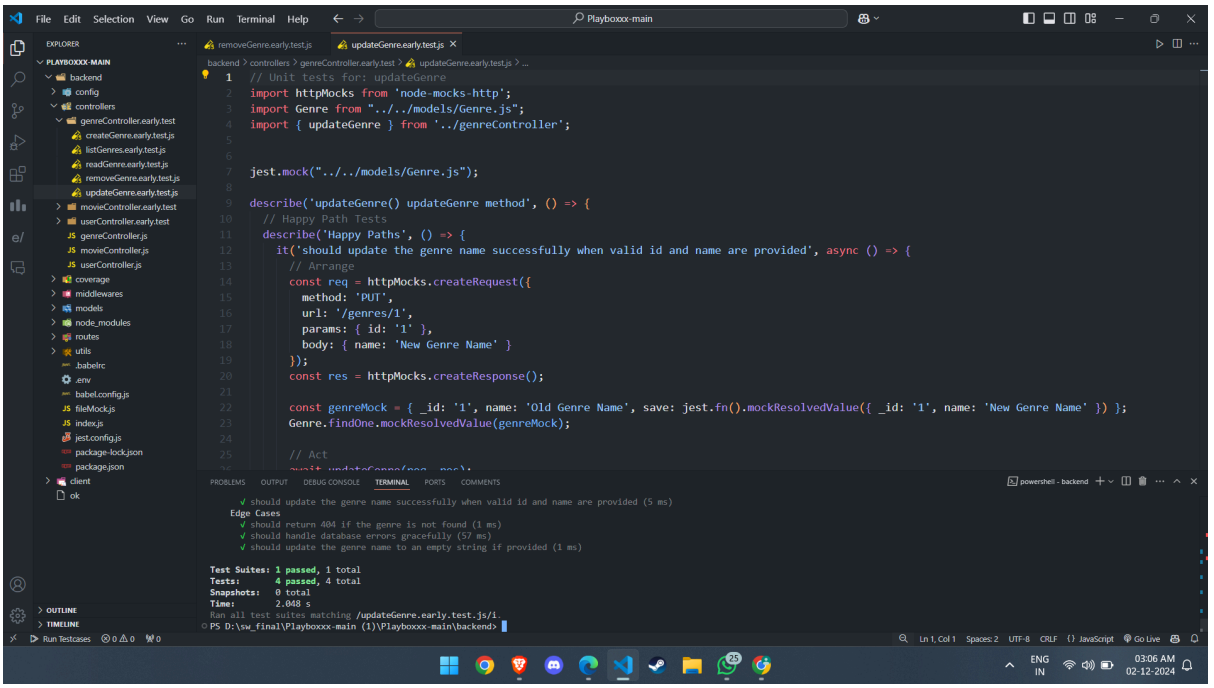
Read genres testing



Remove genres testing

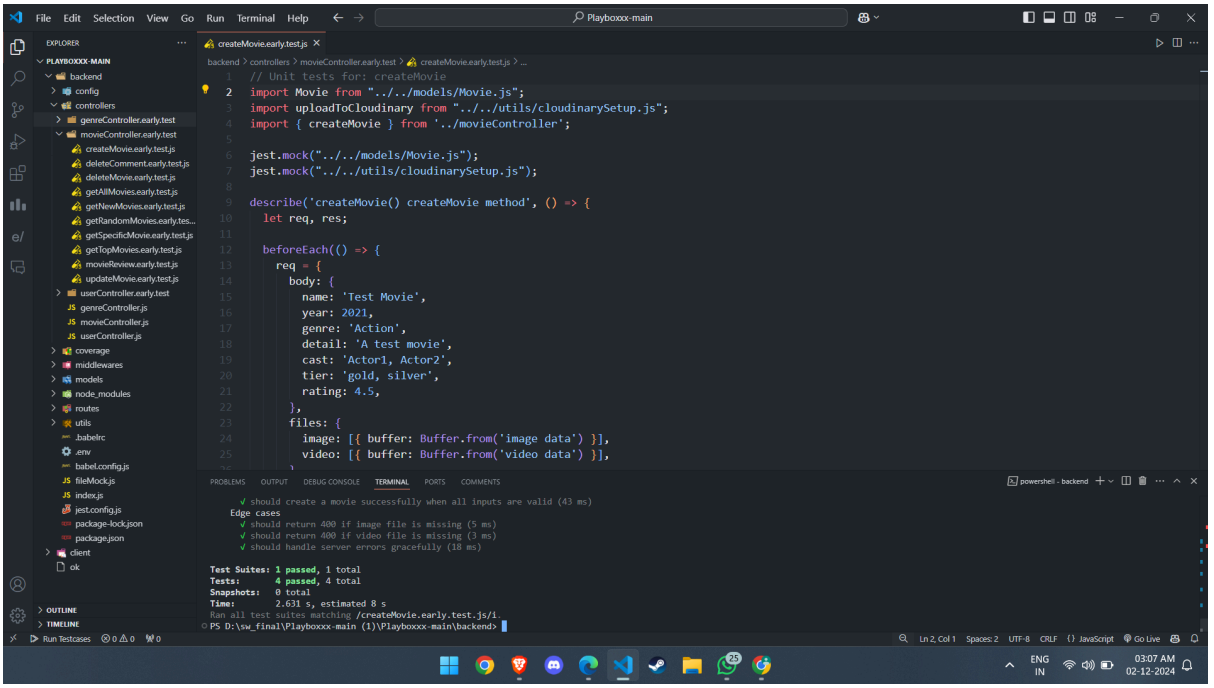


Update genres testing

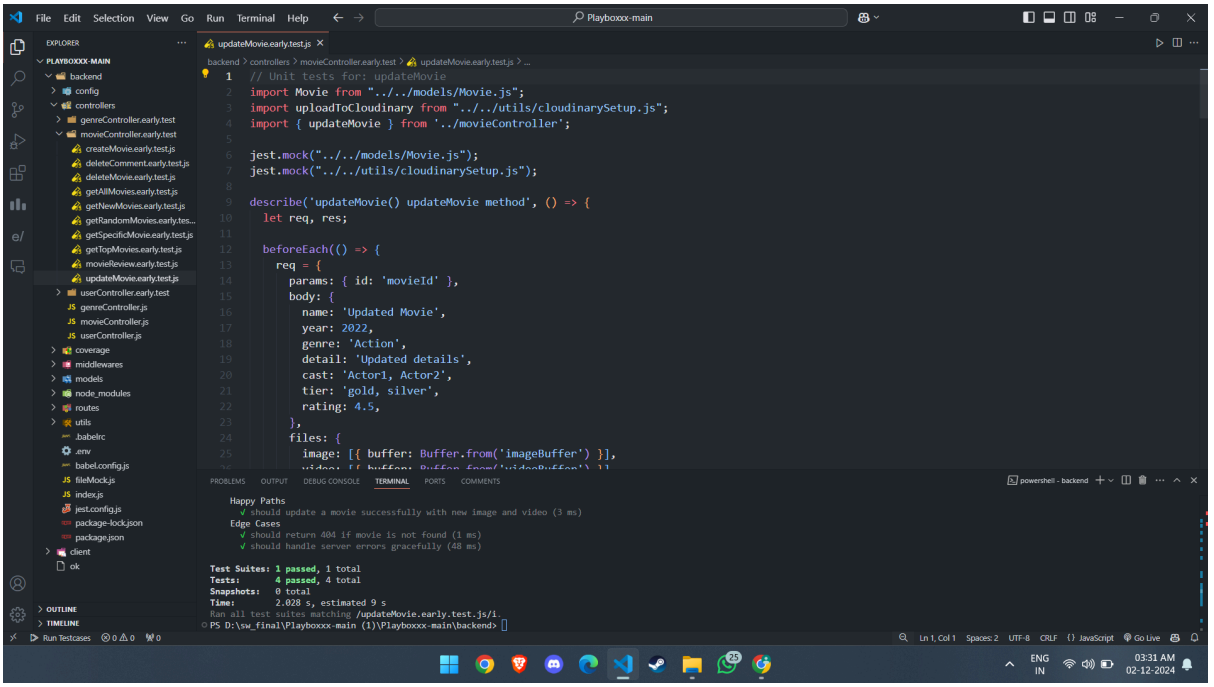


2. Testing of Movie Controller

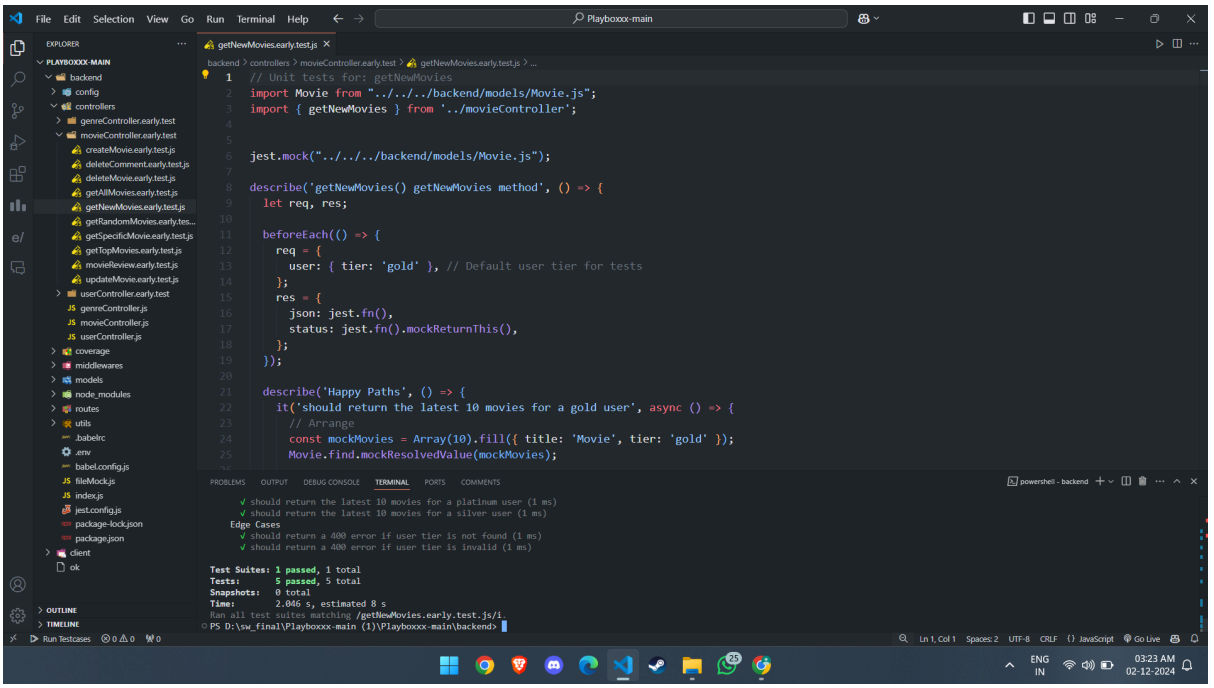
Create Movie testing



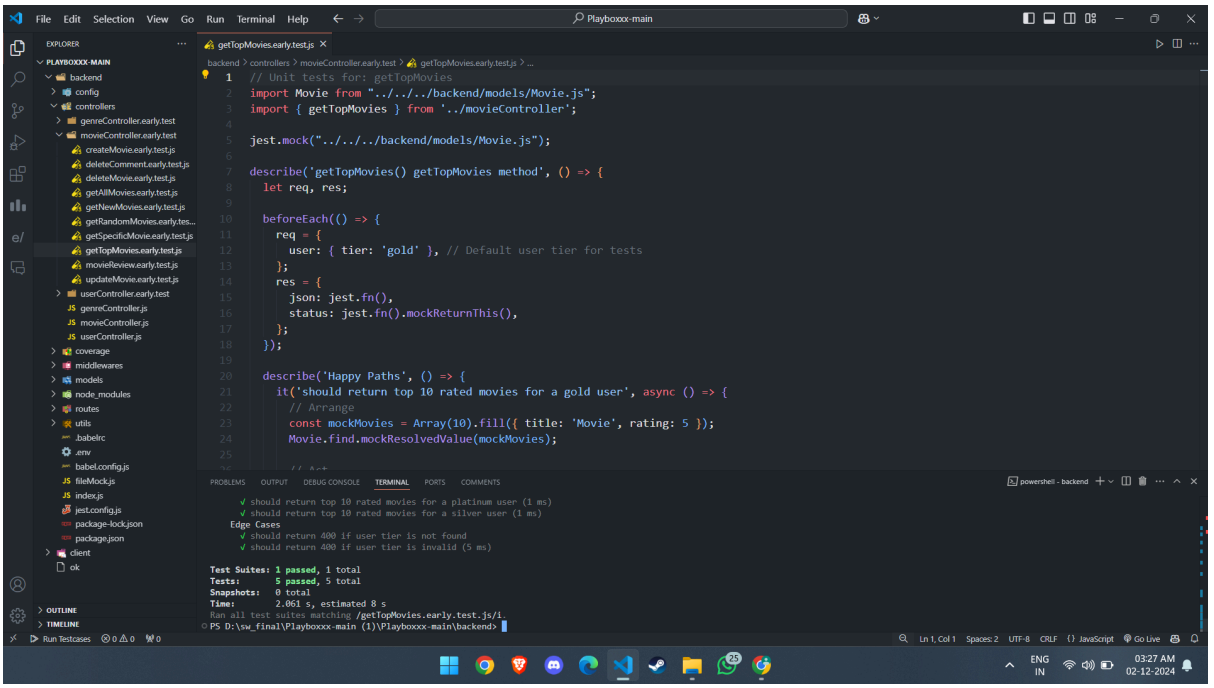
Update Movie testing



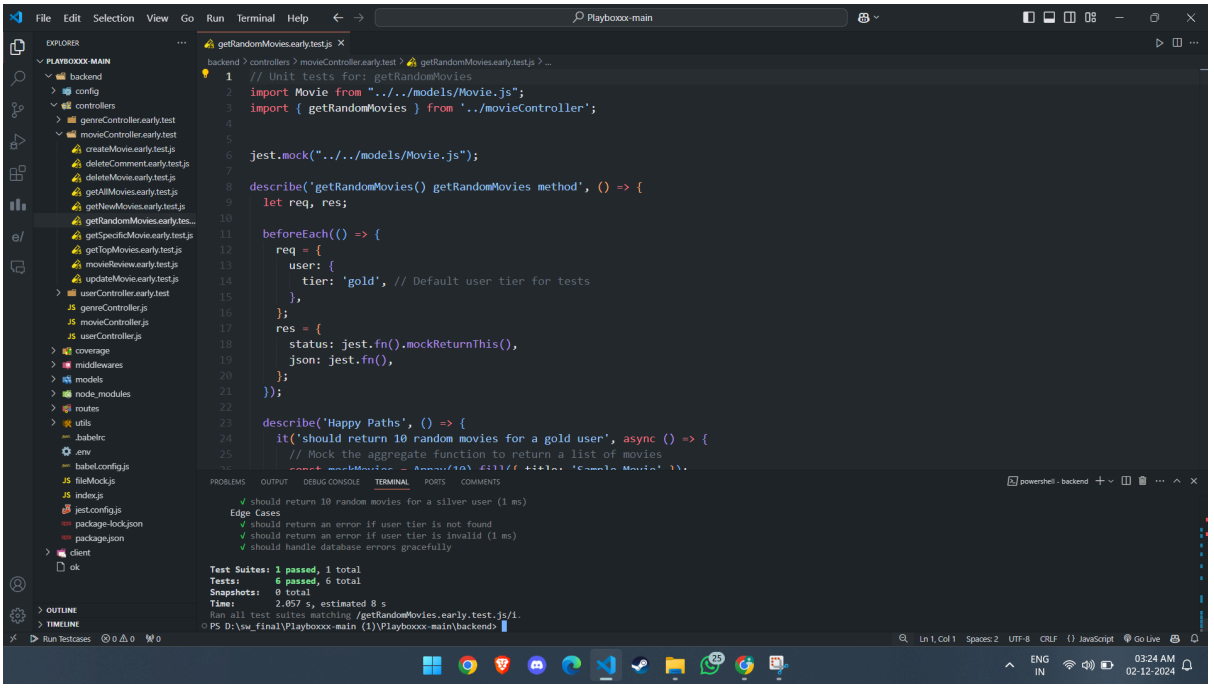
New Movie testing



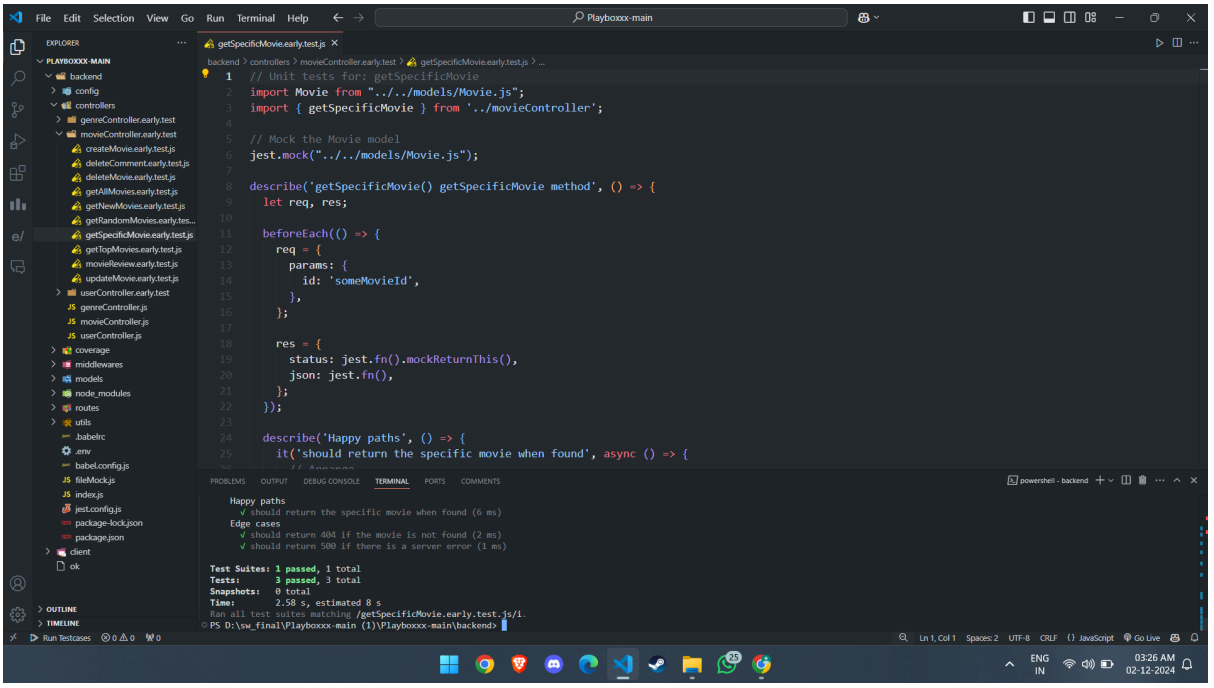
Top Movie testing



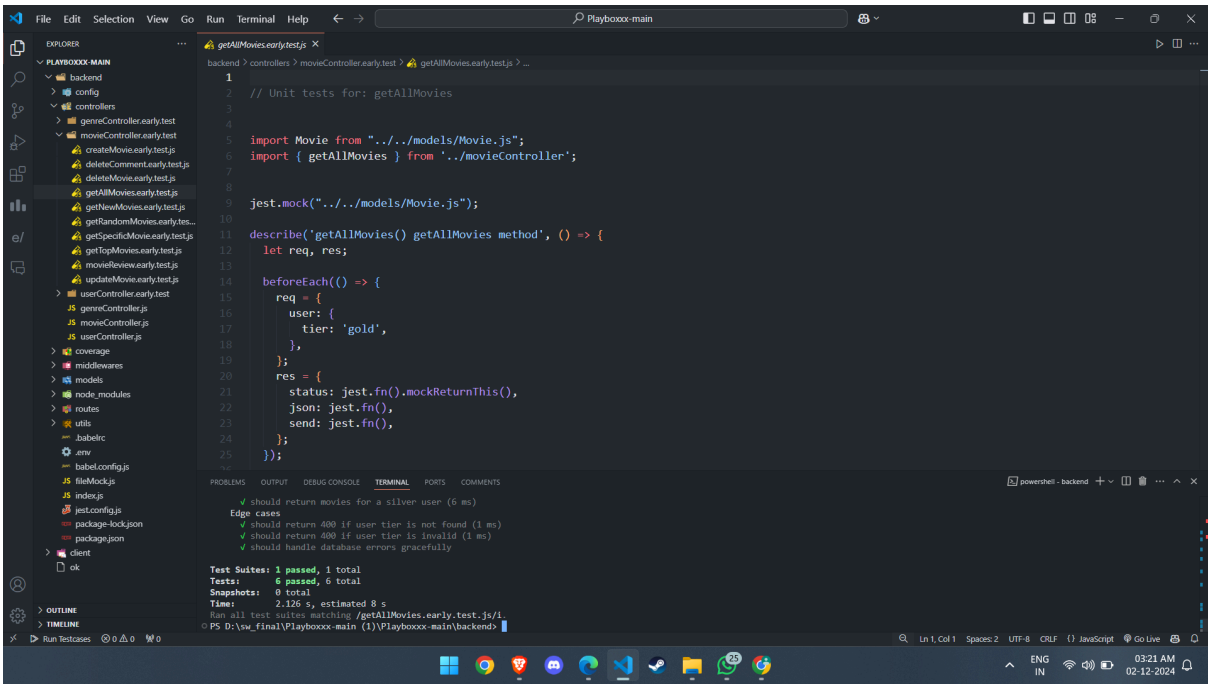
Random Movie testing



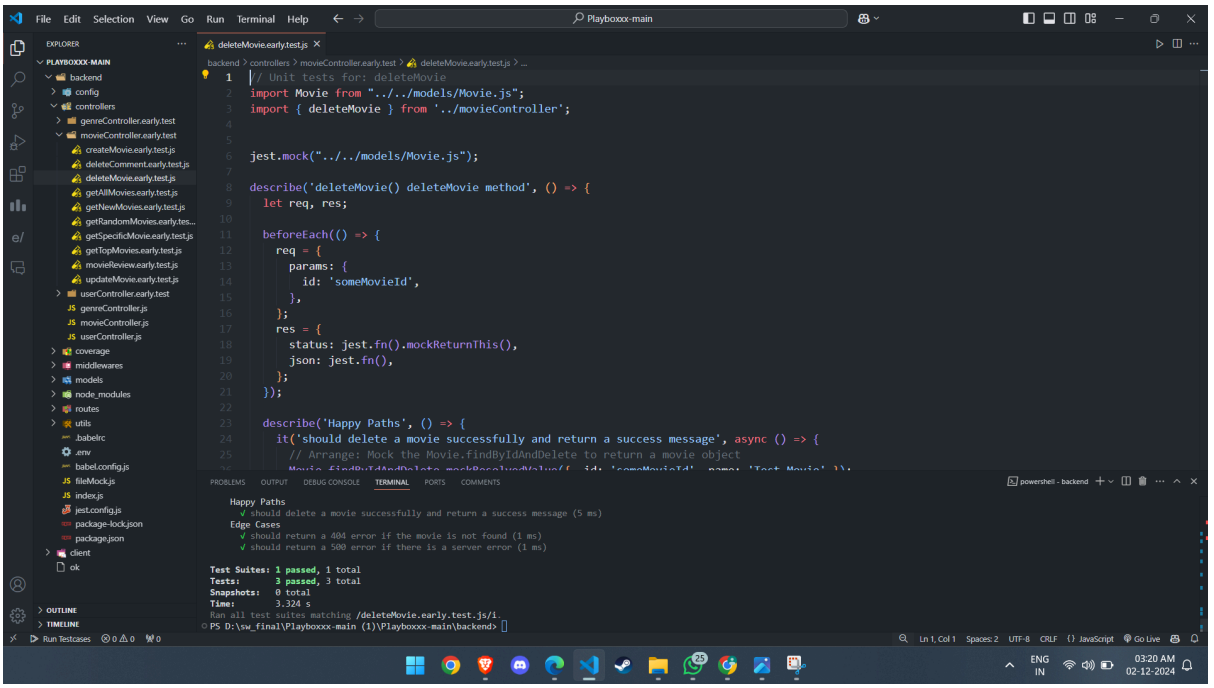
Specific Movie testing



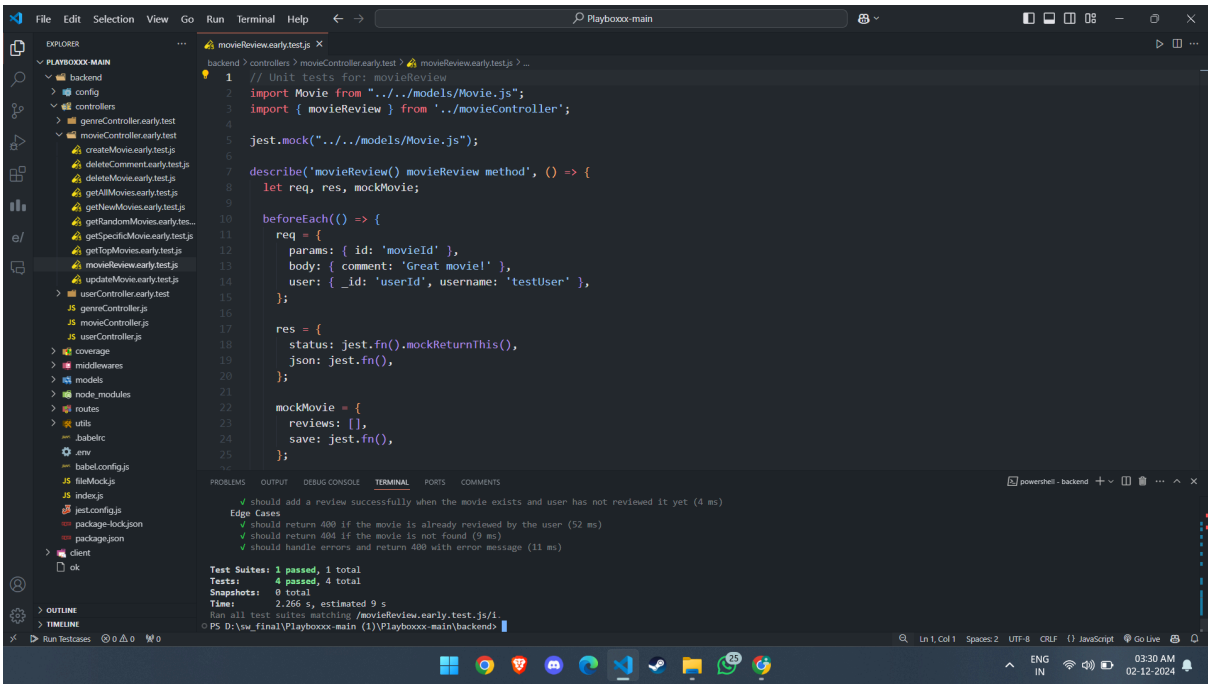
All Movie testing



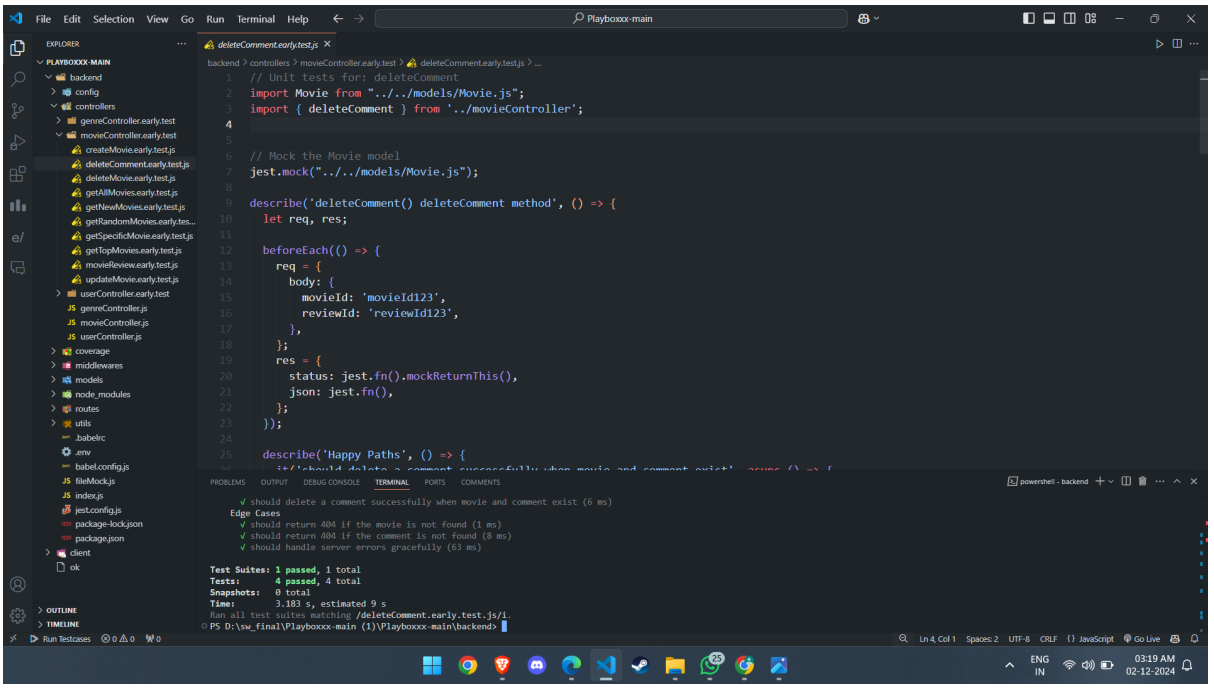
Delete Movie testing



Movie Review testing

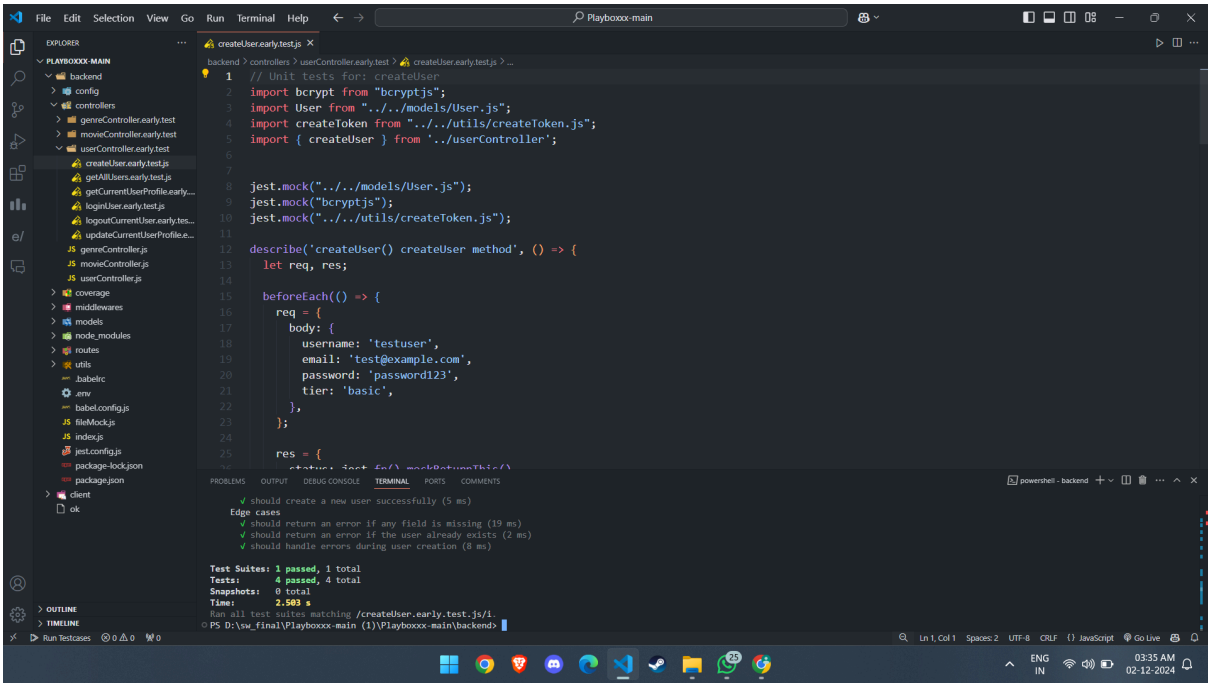


Delete Movie Review testing

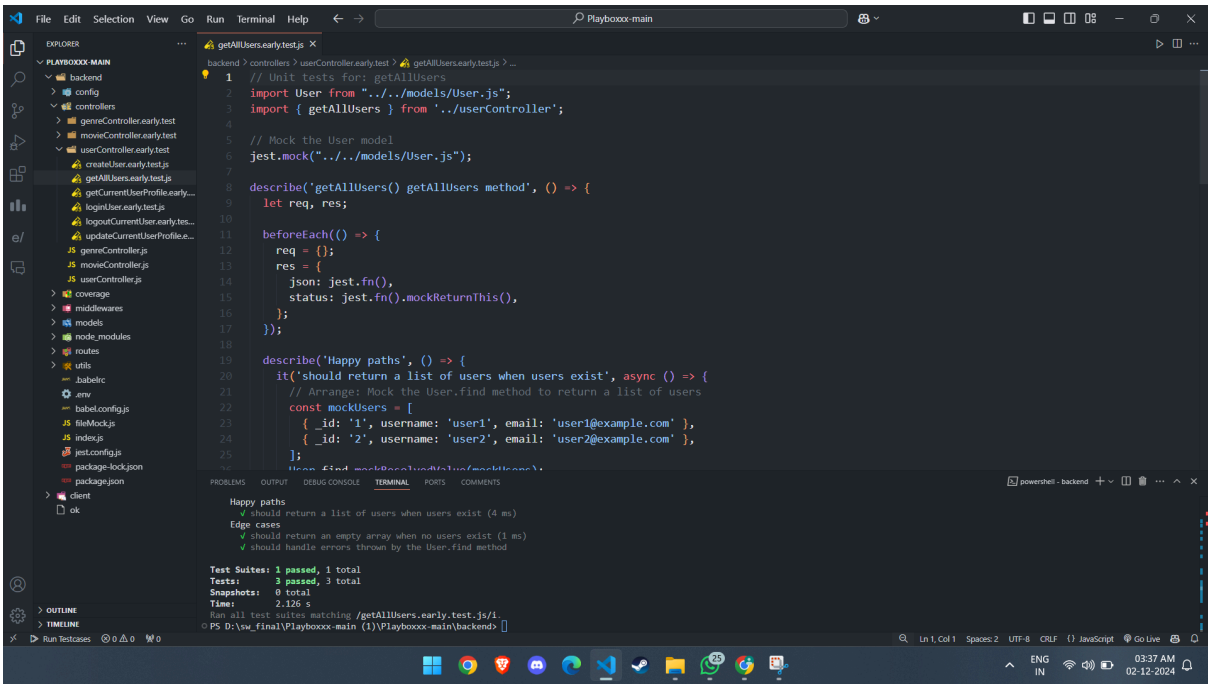


3. Testing of User Controller

Create User testing



Get All User testing



User Profile testing

```
// Unit tests for: getCurrentUserProfile
import User from "../../models/User.js";
import { getCurrentUserProfile } from '../userController';


jest.mock("../../models/User.js");

describe('getCurrentUserProfile() getCurrentUserProfile method', () => {
  let req, res, user;

  beforeEach(() => {
    // Mock request and response objects
    req = {
      user: {
        _id: 'user123',
      },
    };
    res = {
      json: jest.fn(),
      status: jest.fn().mockReturnThis(),
    };

    // Mock user data
    user = {
      _id: 'user123',
```

```
getCurrentUserProfile() getCurrentUserProfile method
  Happy Paths
    √ should return the user profile when the user is found (3 ms)
  Edge Cases
    √ should return 404 status and error message when the user is not found (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.891 s, estimated 2 s
Ran all test suites matching /getCurrentUserProfile.early.test.js/i.
PS D:\sw_final\Playboxxx-main (1)\Playboxxx-main\backend>
```

Login User testing

```
// Unit tests for: loginUser
import bcrypt from "bcryptjs";
import User from "../../models/User.js";
import createToken from "../../utils/createToken.js";
import { loginUser } from '../userController';


jest.mock("../../models/User.js");
jest.mock("bcryptjs");
jest.mock("../../utils/createToken.js");

describe('loginUser() loginUser method', () => {
  let req, res, jsonMock, statusMock;

  beforeEach(() => {
    jsonMock = jest.fn();
    statusMock = jest.fn(() => ({ json: jsonMock }));
    res = { status: statusMock };
  });

  describe('Happy Paths', () => {
    it('should log in a user with valid credentials', async () => {
      // Arrange
      req = { body: { email: 'test@example.com', password: 'password123' } };
      const user = {
```

```
    √ should log in a user with valid credentials (3 ms)
  Edge Cases
    √ should return 401 if user is not found (1 ms)
    √ should return 401 if password is invalid (1 ms)
    √ should handle errors thrown during execution

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.9 s
Ran all test suites matching /loginUser.early.test.js/i.
PS D:\sw_final\Playboxxx-main (1)\Playboxxx-main\backend>
```

❖ Conclusion:

Unit testing is crucial to ensure the quality, maintainability, and reliability of the project. Using Jest, and Istanbul, comprehensive testing was performed to effectively cover back-end and front-end functionalities.