

# 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, and 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

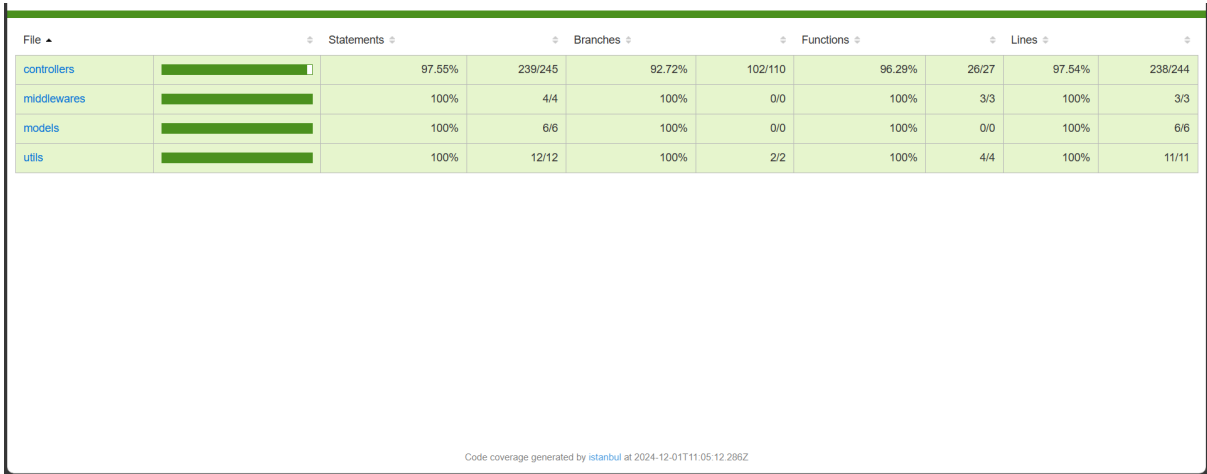


Figure 1: Overall coverage for backend testing.

## Coverage of Controllers

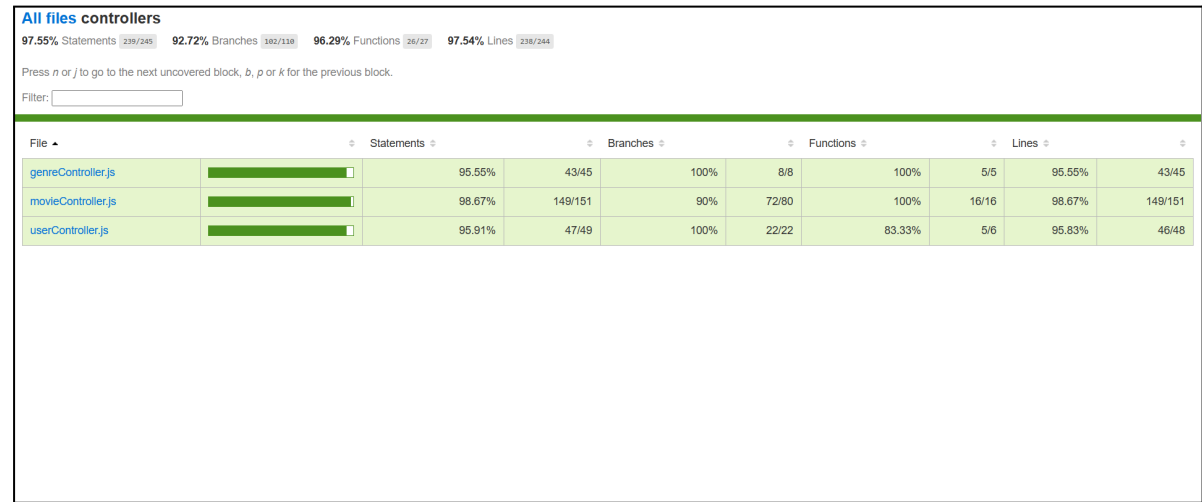
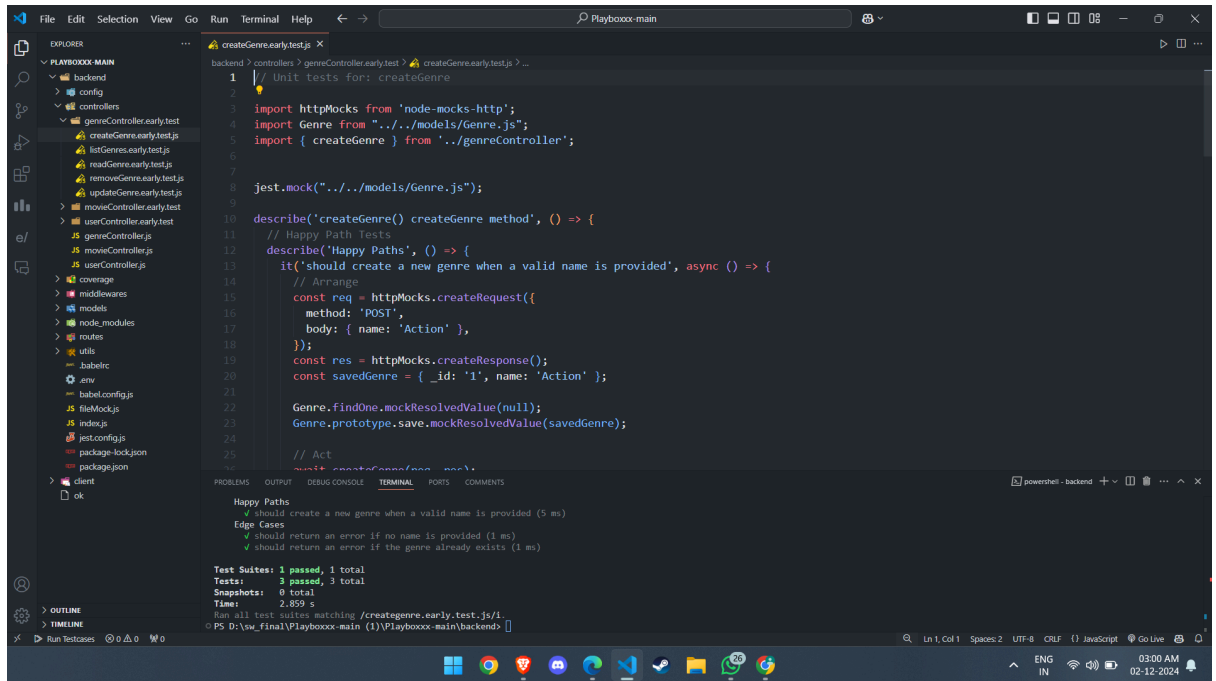


Figure 2: Coverage of backend controllers.

# 1. Testing of genre Controller

## Create genres testing



The screenshot shows a VS Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with folders like 'backend', 'config', 'controllers', 'models', 'middlewares', 'node\_modules', 'routes', 'utils', and 'client'. The code editor displays the file 'createGenre.early.test.js' with the following content:

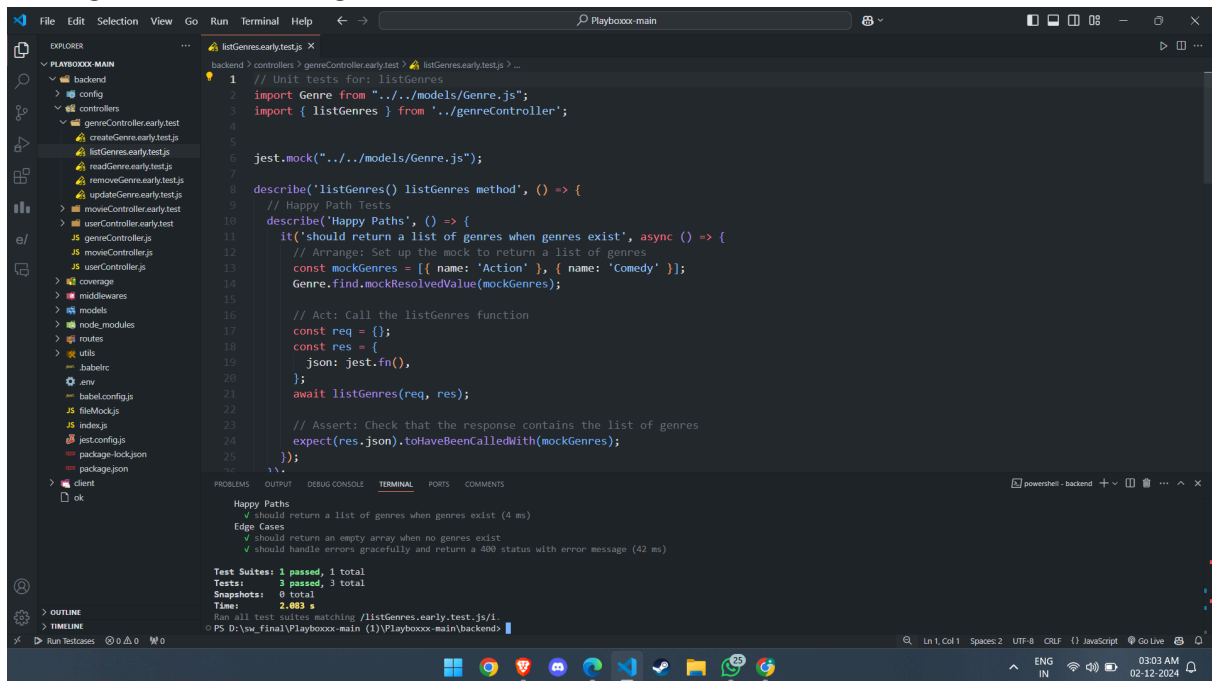
```
1 // Unit tests for: createGenre
2
3 import httpMocks from 'node-mocks-http';
4 import Genre from '../models/Genre.js';
5 import { createGenre } from '../genreController';
6
7
8 jest.mock("../../models/Genre.js");
9
10 describe('createGenre() createGenre method', () => {
11   // Happy Path Tests
12   describe('Happy Paths', () => {
13     it('should create a new genre when a valid name is provided', async () => {
14       // Arrange
15       const req = httpMocks.createRequest({
16         method: 'POST',
17         body: { name: 'Action' },
18       });
19       const res = httpMocks.createResponse();
20       const savedGenre = { _id: '1', name: 'Action' };
21
22       Genre.findOne.mockResolvedValue(null);
23       Genre.prototype.save.mockResolvedValue(savedGenre);
24
25       // Act
26       await createGenre(req, res);
27     });
28   });
29
30   // Edge Cases
31   describe('Edge Cases', () => {
32     it('should return an error if no name is provided (1 ms)', async () => {
33       // Arrange
34       const req = httpMocks.createRequest({
35         method: 'POST',
36         body: { },
37       });
38       const res = httpMocks.createResponse();
39
40       Genre.findOne.mockResolvedValue(null);
41       Genre.prototype.save.mockResolvedValue(savedGenre);
42
43       // Act
44       await createGenre(req, res);
45     });
46     it('should return an error if the genre already exists (1 ms)', async () => {
47       // Arrange
48       const req = httpMocks.createRequest({
49         method: 'POST',
50         body: { name: 'Action' },
51       });
52       const res = httpMocks.createResponse();
53       const savedGenre = { _id: '1', name: 'Action' };
54
55       Genre.findOne.mockResolvedValue(savedGenre);
56       Genre.prototype.save.mockResolvedValue(savedGenre);
57
58       // Act
59       await createGenre(req, res);
60     });
61   });
62 });
```

The bottom panel shows the test results for 'createGenre.early.test.js':

- Happy Paths
  - ✓ should create a new genre when a valid name is provided (5 ms)
- Edge Cases
  - ✓ should return an error if no name is provided (1 ms)
  - ✓ should return an error if the genre already exists (1 ms)

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 2.859 s  
Run all test suites matching /createGenre.early.test.js/

## List genres testing



The screenshot shows a VS Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with folders like 'backend', 'config', 'controllers', 'models', 'middlewares', 'node\_modules', 'routes', 'utils', and 'client'. The code editor displays the file 'listGenres.early.test.js' with the following content:

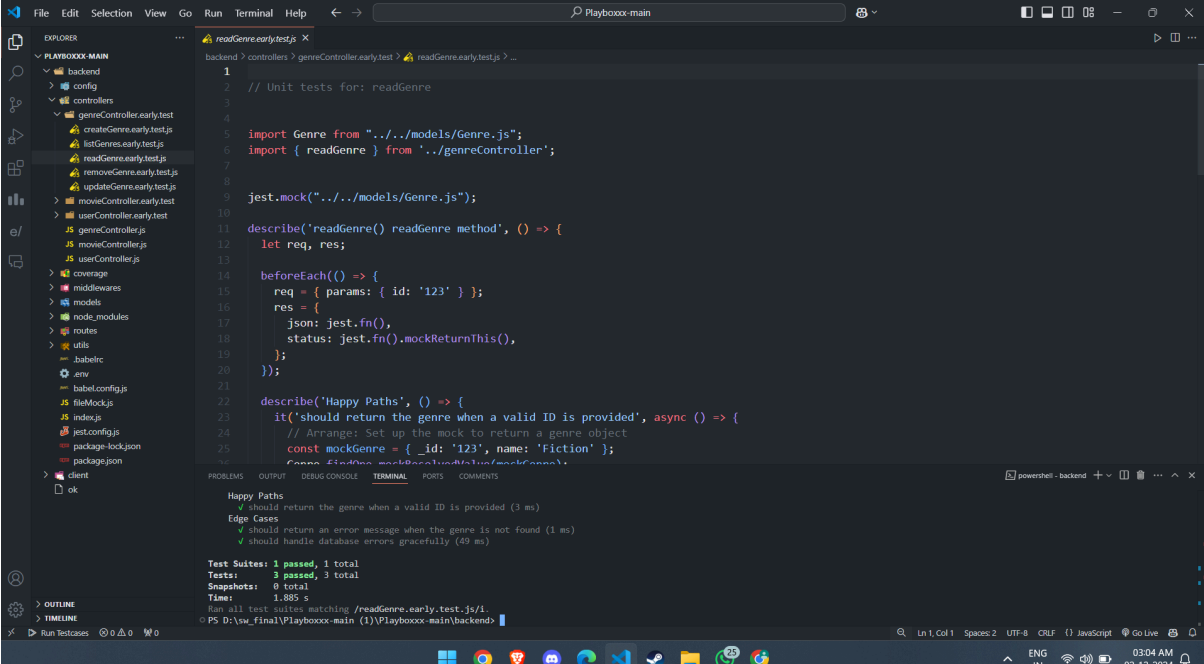
```
1 // Unit tests for: listGenres
2
3 import Genre from '../models/Genre.js';
4 import { listGenres } from '../genreController';
5
6
7 jest.mock("../../models/Genre.js");
8
9 describe('listGenres() listGenres method', () => {
10   // Happy Path Tests
11   describe('Happy Paths', () => {
12     it('should return a list of genres when genres exist', async () => {
13       // Arrange: Set up the mock to return a list of genres
14       const mockGenres = [{ name: 'Action' }, { name: 'Comedy' }];
15       Genre.find.mockResolvedValue(mockGenres);
16
17       // Act: Call the listGenres function
18       const req = {};
19       const res = {
20         json: jest.fn(),
21       };
22       await listGenres(req, res);
23
24       // Assert: Check that the response contains the list of genres
25       expect(res.json).toHaveBeenCalledWith(mockGenres);
26     });
27   });
28
29   // Edge Cases
30   describe('Edge Cases', () => {
31     it('should return an empty array when no genres exist', async () => {
32       // Arrange
33       const req = {};
34       const res = {
35         json: jest.fn(),
36       };
37       Genre.find.mockResolvedValue([]);
38
39       // Act
40       await listGenres(req, res);
41
42       // Assert
43       expect(res.json).toHaveBeenCalledWith([]);
44     });
45     it('should handle errors gracefully and return a 400 status with error message (42 ms)', async () => {
46       // Arrange
47       const req = {};
48       const res = {
49         json: jest.fn(),
50       };
51       Genre.find.mockRejectedValue(new Error('Database error'));
52
53       // Act
54       await listGenres(req, res);
55
56       // Assert
57       expect(res.status).toBe(400);
58       expect(res.json).toHaveBeenCalledWith('Database error');
59     });
60   });
61 });
```

The bottom panel shows the test results for 'listGenres.early.test.js':

- Happy Paths
  - ✓ should return a list of genres when genres exist (4 ms)
- Edge Cases
  - ✓ should return an empty array when no genres exist
  - ✓ should handle errors gracefully and return a 400 status with error message (42 ms)

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 2.883 s  
Run all test suites matching /listGenres.early.test.js/

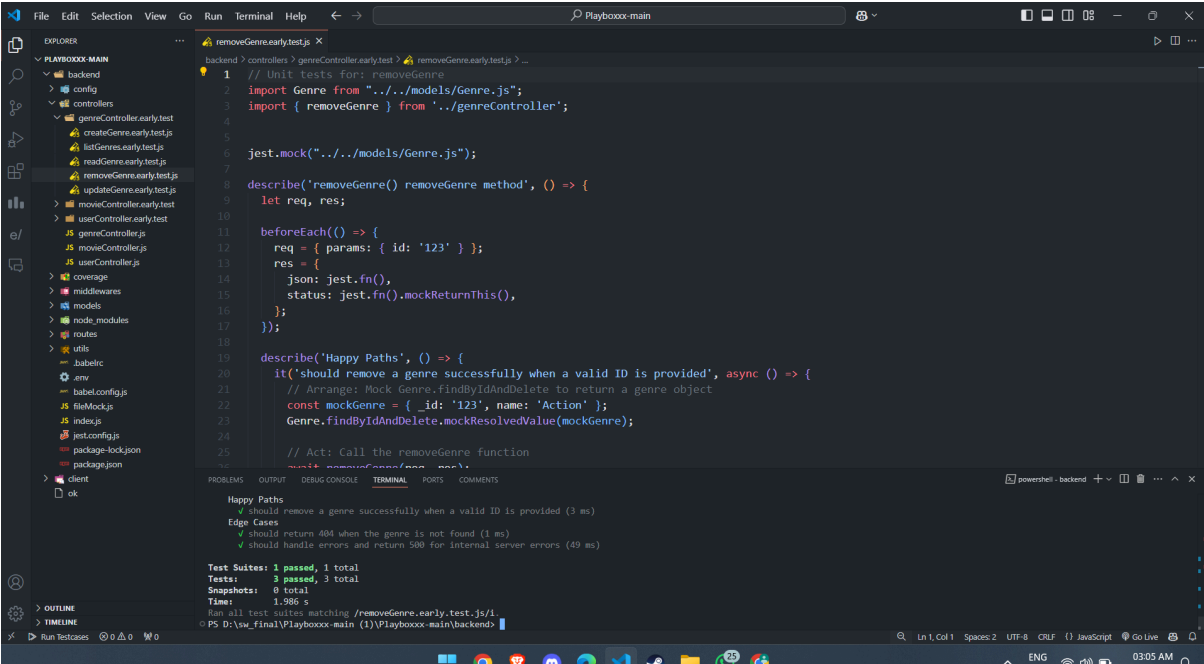
## Read genres testing



```
1 // Unit tests for: readGenre
2
3
4
5 import Genre from "../../models/Genre.js";
6 import { readGenre } from '../genreController';
7
8
9 jest.mock("../../models/Genre.js");
10
11 describe('readGenre() readGenre method', () => {
12   let req, res;
13
14   beforeEach(() => {
15     req = { params: { id: '123' } };
16     res = {
17       json: jest.fn(),
18       status: jest.fn().mockReturnThis(),
19     };
20   });
21
22   describe('Happy Paths', () => {
23     it('should return the genre when a valid ID is provided', async () => {
24       // Arrange: Set up the mock to return a genre object
25       const mockGenre = { _id: '123', name: 'Fiction' };
26       Genre.findByIdAndDelete.mockResolvedValue(mockGenre);
27     });
28   });
29
30   describe('Edge Cases', () => {
31     it('should return an error message when the genre is not found (1 ms)', async () => {
32       Genre.findByIdAndDelete.mockResolvedValue(null);
33     });
34     it('should handle database errors gracefully (49 ms)', async () => {
35       Genre.findByIdAndDelete.mockRejectedValue(new Error('Database error'));
36     });
37   });
38 });
```

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 1.685 s  
Ran all test suites matching /readGenre.early.test.js/

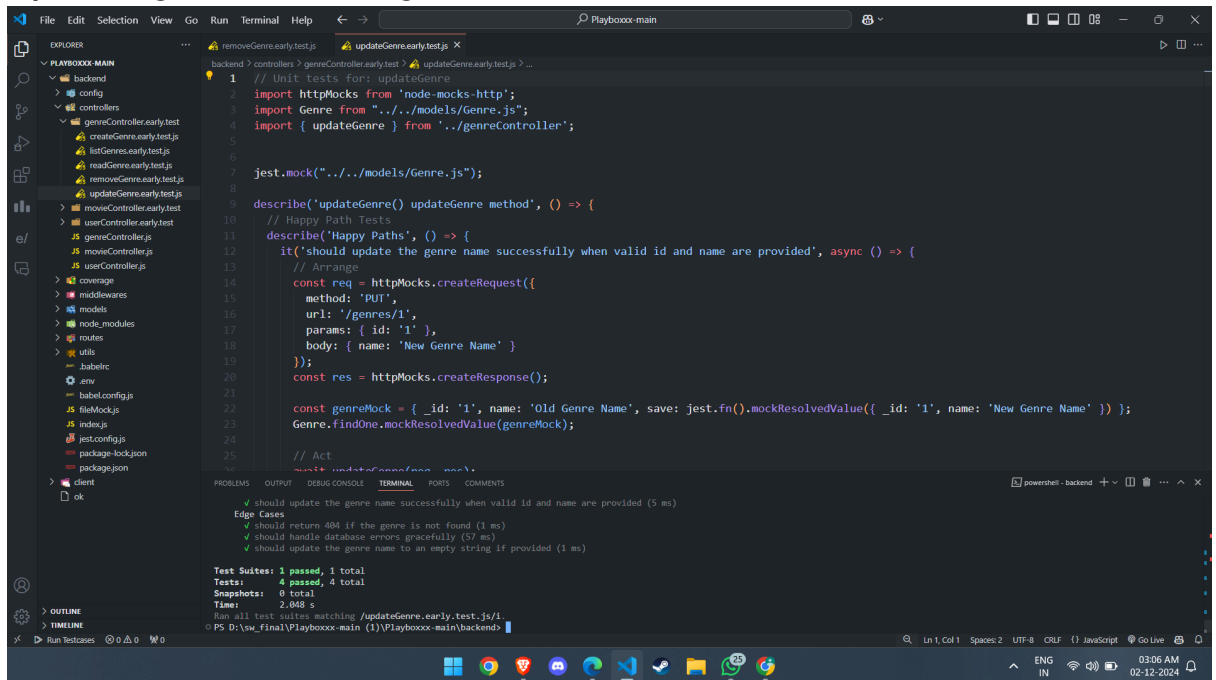
## Remove genres testing



```
1 // Unit tests for: removeGenre
2
3 import Genre from "../../models/Genre.js";
4 import { removeGenre } from '../genreController';
5
6
7 jest.mock("../../models/Genre.js");
8
9 describe('removeGenre() removeGenre method', () => {
10   let req, res;
11
12   beforeEach(() => {
13     req = { params: { id: '123' } };
14     res = {
15       json: jest.fn(),
16       status: jest.fn().mockReturnThis(),
17     };
18   });
19
20   describe('Happy Paths', () => {
21     it('should remove a genre successfully when a valid ID is provided', async () => {
22       // Arrange: Mock Genre.findByIdAndDelete to return a genre object
23       const mockGenre = { _id: '123', name: 'Action' };
24       Genre.findByIdAndDelete.mockResolvedValue(mockGenre);
25
26       // Act: Call the removeGenre function
27       await removeGenre(req, res);
28     });
29   });
30
31   describe('Edge Cases', () => {
32     it('should return 404 when the genre is not found (1 ms)', async () => {
33       Genre.findByIdAndDelete.mockResolvedValue(null);
34     });
35     it('should handle errors and return 500 for internal server errors (49 ms)', async () => {
36       Genre.findByIdAndDelete.mockRejectedValue(new Error('Database error'));
37     });
38   });
39 });
```

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 1.986 s  
Ran all test suites matching /removeGenre.early.test.js/

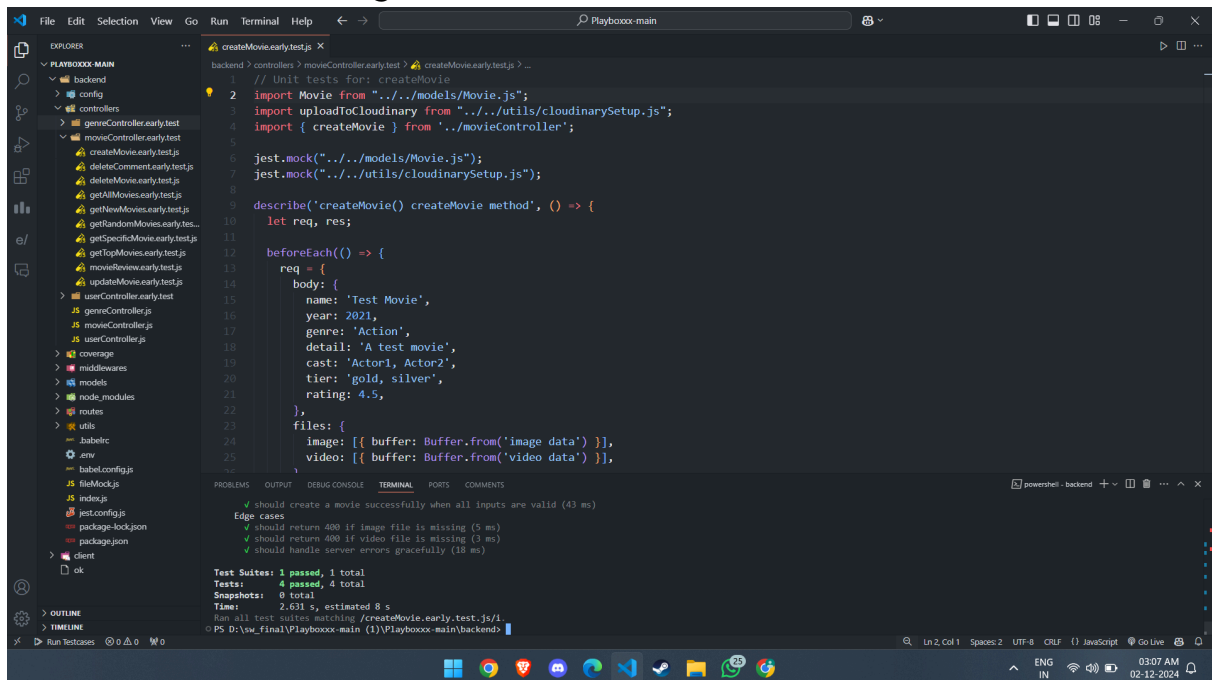
## Update genres testing



```
1 // Unit tests for: updateGenre
2 import httpMocks from 'node-mocks-http';
3 import Genre from '../models/Genre.js';
4 import { updateGenre } from '../genreController';
5
6
7 jest.mock('../models/Genre.js');
8
9 describe('updateGenre() updateGenre method', () => {
10   // Happy Path Tests
11   describe('Happy Paths', () => {
12     it('should update the genre name successfully when valid id and name are provided', async () => {
13       // Arrange
14       const req = httpMocks.createRequest({
15         method: 'PUT',
16         url: '/genres/1',
17         params: { id: '1' },
18         body: { name: 'New Genre Name' }
19       });
20       const res = httpMocks.createResponse();
21
22       const genreMock = { id: '1', name: 'Old Genre Name', save: jest.fn().mockResolvedValue({ id: '1', name: 'New Genre Name' }) };
23       Genre.findOne.mockResolvedValue(genreMock);
24
25       // Act
26       const updateGenre = jest.requireActual('../genreController').updateGenre;
27       await updateGenre(req, res);
28
29       // Assert
30       expect(res.statusCode).toBe(200);
31       expect(res._getJSONData().name).toBe('New Genre Name');
32       expect(genreMock.save).toHaveBeenCalled();
33     });
34
35     // Edge Cases
36     it('should return 404 if the genre is not found (1 ms)', async () => {
37       const req = httpMocks.createRequest({ method: 'PUT', url: '/genres/999' });
38       const res = httpMocks.createResponse();
39       Genre.findOne.mockResolvedValue(null);
40       await updateGenre(req, res);
41       expect(res.statusCode).toBe(404);
42     });
43
44     it('should handle database errors gracefully (57 ms)', async () => {
45       const req = httpMocks.createRequest({ method: 'PUT', url: '/genres/1' });
46       const res = httpMocks.createResponse();
47       Genre.findOne.mockRejectedValue(new Error('Database error'));
48       await updateGenre(req, res);
49       expect(res.statusCode).toBe(500);
50     });
51
44     it('should update the genre name to an empty string if provided (1 ms)', async () => {
52       const req = httpMocks.createRequest({ method: 'PUT', url: '/genres/1', body: { name: '' } });
53       const res = httpMocks.createResponse();
54       Genre.findOne.mockResolvedValue({ id: '1', name: 'Old Genre Name' });
55       await updateGenre(req, res);
56       expect(res._getJSONData().name).toBe('');
57     });
58   });
59 });
60
61 Test Suites: 1 passed, 1 total
62 Tests: 4 passed, 4 total
63 Snapshots: 0 total
64 Time: 2.048 s
65 Ran all test suites matching /updateGenre.early.test.js/
66 PS D:\sw_final\Playbox-main (1)\Playbox-main\backend>
```

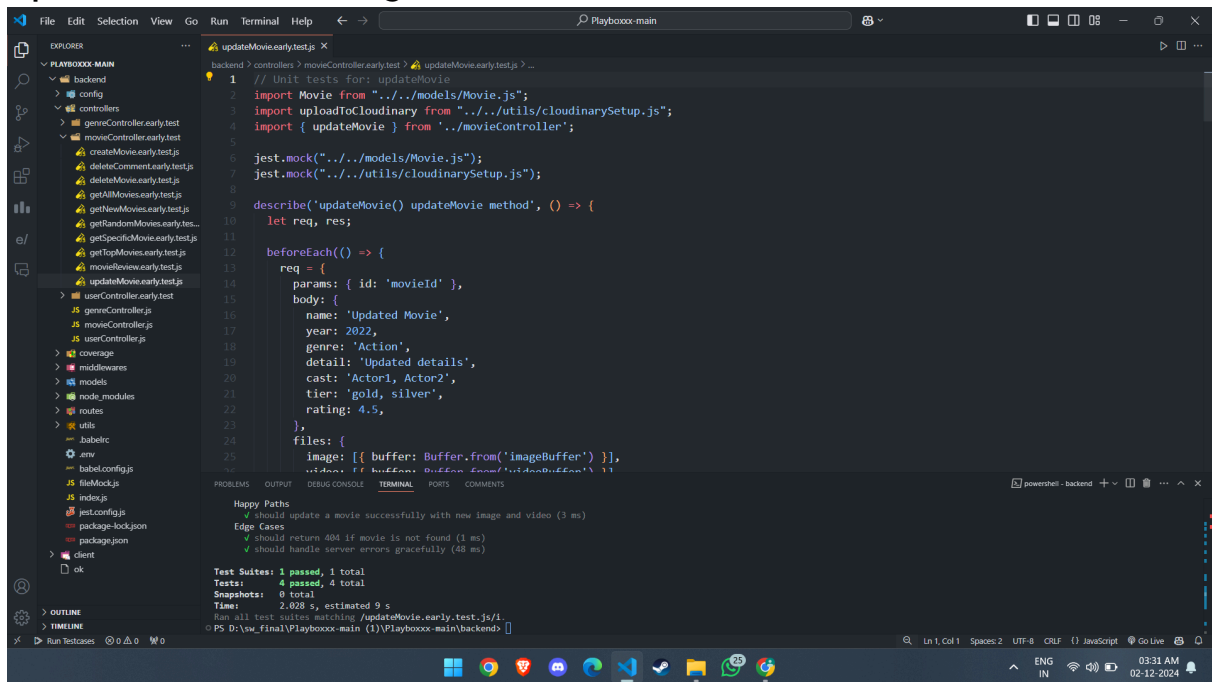
## 2. Testing of Movie Controller

### Create Movie testing



```
1 // Unit tests for: createMovie
2 import Movie from '../models/Movie.js';
3 import uploadToCloudinary from '../utils/cloudinarySetup.js';
4 import { createMovie } from '../movieController';
5
6
7 jest.mock('../models/Movie.js');
8 jest.mock('../utils/cloudinarySetup.js');
9
10 describe('createMovie() createMovie method', () => {
11   let req, res;
12
13   beforeEach(() => {
14     req = {
15       body: {
16         name: 'Test Movie',
17         year: 2021,
18         genre: 'Action',
19         detail: 'A test movie',
20         cast: 'Actor1, Actor2',
21         tier: 'gold, silver',
22         rating: 4.5,
23       },
24       files: {
25         image: [{ buffer: Buffer.from('image data') }],
26         video: [{ buffer: Buffer.from('video data') }],
27       },
28     };
29     res = httpMocks.createResponse();
30   });
31
32   it('should create a movie successfully when all inputs are valid (43 ms)', async () => {
33     const createMovie = jest.requireActual('../movieController').createMovie;
34     await createMovie(req, res);
35     expect(res.statusCode).toBe(201);
36     expect(res._getJSONData().name).toBe('Test Movie');
37     expect(res._getJSONData().year).toBe(2021);
38     expect(res._getJSONData().genre).toBe('Action');
39     expect(res._getJSONData().detail).toBe('A test movie');
40     expect(res._getJSONData().cast).toBe('Actor1, Actor2');
41     expect(res._getJSONData().tier).toBe('gold, silver');
42     expect(res._getJSONData().rating).toBe(4.5);
43     expect(res._getJSONData().image).toBe('image data');
44     expect(res._getJSONData().video).toBe('video data');
45   });
46
47   // Edge cases
48   it('should return 400 if image file is missing (5 ms)', async () => {
49     const req = { body: { name: 'Test Movie', year: 2021, genre: 'Action', detail: 'A test movie', cast: 'Actor1, Actor2', tier: 'gold, silver', rating: 4.5 }, files: {} };
50     const res = httpMocks.createResponse();
51     const createMovie = jest.requireActual('../movieController').createMovie;
52     await createMovie(req, res);
53     expect(res.statusCode).toBe(400);
54   });
55
56   it('should return 400 if video file is missing (3 ms)', async () => {
57     const req = { body: { name: 'Test Movie', year: 2021, genre: 'Action', detail: 'A test movie', cast: 'Actor1, Actor2', tier: 'gold, silver', rating: 4.5 }, files: { image: [{ buffer: Buffer.from('image data') }] } };
58     const res = httpMocks.createResponse();
59     const createMovie = jest.requireActual('../movieController').createMovie;
60     await createMovie(req, res);
61     expect(res.statusCode).toBe(400);
62   });
63
64   it('should handle server errors gracefully (18 ms)', async () => {
65     const req = { body: { name: 'Test Movie', year: 2021, genre: 'Action', detail: 'A test movie', cast: 'Actor1, Actor2', tier: 'gold, silver', rating: 4.5 }, files: { image: [{ buffer: Buffer.from('image data') }], video: [{ buffer: Buffer.from('video data') }] } };
66     const res = httpMocks.createResponse();
67     const createMovie = jest.requireActual('../movieController').createMovie;
68     await createMovie(req, res);
69     expect(res.statusCode).toBe(500);
70   });
71 });
72
73 Test Suites: 1 passed, 1 total
74 Tests: 4 passed, 4 total
75 Snapshots: 0 total
76 Time: 2.631 s, estimated 8 s
77 Ran all test suites matching /createMovie.early.test.js/
78 PS D:\sw_final\Playbox-main (1)\Playbox-main\backend>
```

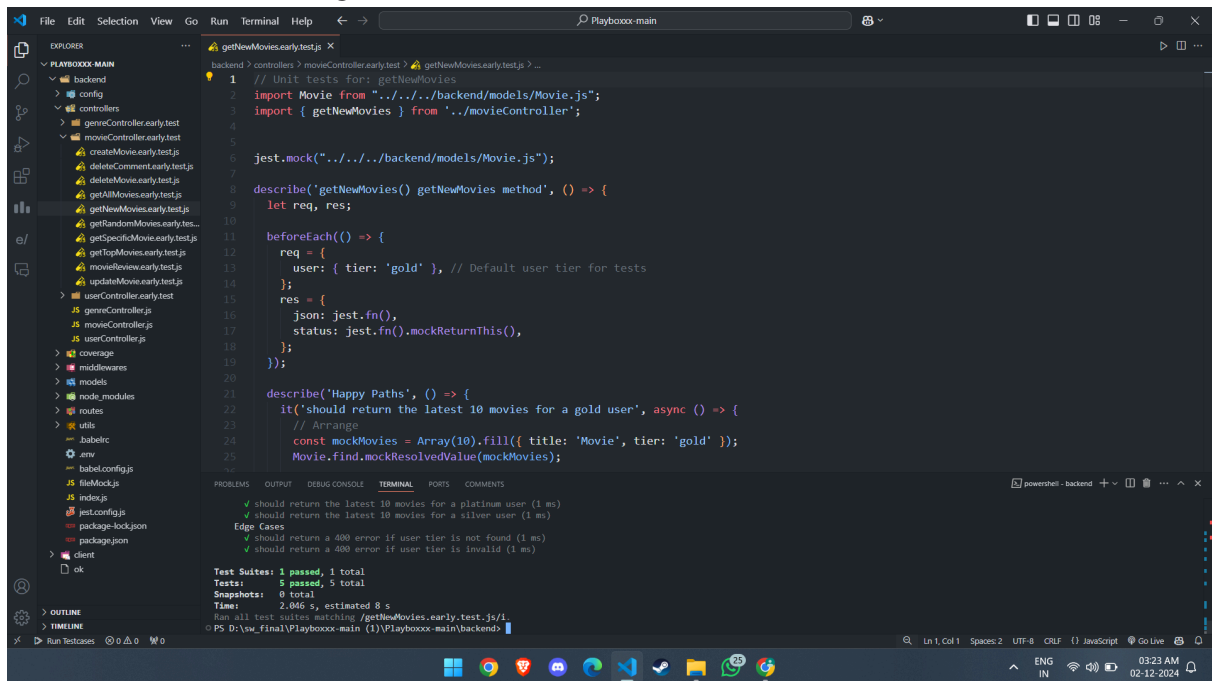
# Update Movie testing



```
1 // Unit tests for: updateMovie
2 import Movie from "../../models/Movie.js";
3 import uploadToCloudinary from "../../utils/cloudinarySetup.js";
4 import { updateMovie } from "../movieController";
5
6 jest.mock("../../models/Movie.js");
7 jest.mock("../../utils/cloudinarySetup.js");
8
9 describe('updateMovie() updateMovie method', () => {
10   let req, res;
11
12   beforeEach(() => {
13     req = {
14       params: { id: 'movieId' },
15       body: {
16         name: 'Updated Movie',
17         year: 2022,
18         genre: 'Action',
19         detail: 'Updated details',
20         cast: 'Actor1, Actor2',
21         tier: 'gold, silver',
22         rating: 4.5,
23       },
24       files: {
25         image: [{ buffer: Buffer.from('imageBuffer') }],
26         video: [{ buffer: Buffer.from('videoBuffer') }],
27       }
28     };
29     res = {
30       status: jest.fn().mockReturnThis(),
31       json: jest.fn()
32     };
33   });
34
35   describe('Happy Paths', () => {
36     it('should update a movie successfully with new image and video (3 ms)', () => {
37       // ...
38     });
39   });
40
41   describe('Edge Cases', () => {
42     it('should return 404 if movie is not found (1 ms)', () => {
43       // ...
44     });
45     it('should handle server errors gracefully (48 ms)', () => {
46       // ...
47     });
48   });
49 });
```

Test Suites: 1 passed, 1 total  
Tests: 4 passed, 4 total  
Snapshots: 0 total  
Time: 2.028 s, estimated 9 s  
Ran all test suites matching /updateMovie.early.test.js/i

# New Movie testing



```
1 // Unit tests for: getNewMovies
2 import Movie from "../../models/Movie.js";
3 import { getNewMovies } from "../movieController";
4
5 jest.mock("../../models/Movie.js");
6
7 describe('getNewMovies() getNewMovies method', () => {
8   let req, res;
9
10   beforeEach(() => {
11     req = {
12       user: { tier: 'gold' }, // Default user tier for tests
13     };
14     res = {
15       json: jest.fn(),
16       status: jest.fn().mockReturnThis(),
17     };
18   });
19
20   describe('Happy Paths', () => {
21     it('should return the latest 10 movies for a gold user, async () => {
22       // Arrange
23       const mockMovies = Array(10).fill({ title: 'Movie', tier: 'gold' });
24       const mockResolvedValue = Promise.resolve(mockMovies);
25       jest.mock('Movie', () => mockResolvedValue);
26
27       // Act
28       getNewMovies(req, res);
29
30       // Assert
31       expect(res.status).toHaveBeenCalledWith(200);
32       expect(res.json).toHaveBeenCalledWith(mockMovies);
33     });
34   });
35
36   describe('Edge Cases', () => {
37     it('should return a 400 error if user tier is not found (1 ms)', () => {
38       // ...
39     });
40     it('should return a 400 error if user tier is invalid (1 ms)', () => {
41       // ...
42     });
43   });
44 });
```

Test Suites: 1 passed, 1 total  
Tests: 5 passed, 5 total  
Snapshots: 0 total  
Time: 2.046 s, estimated 8 s  
Ran all test suites matching /getNewMovies.early.test.js/i

The screenshot shows a VS Code editor with a Jest test file named `getTopMovies.early.test.js` open in the main editor. The Explorer sidebar on the left displays the project structure, including folders like `backend`, `controllers`, and `models`. The main editor shows the following code:

```

1 // Unit tests for: getTopMovies
2 import Movie from "../../backend/models/Movie.js";
3 import { getTopMovies } from '../movieController';
4
5 jest.mock("../../backend/models/Movie.js");
6
7 describe('getTopMovies() getTopMovies method', () => {
8   let req, res;
9
10  beforeEach(() => {
11    req = {
12      user: { tier: 'gold' }, // Default user tier for tests
13    };
14    res = {
15      json: jest.fn(),
16      status: jest.fn().mockReturnThis(),
17    };
18  });
19
20  describe('Happy Paths', () => {
21    it('should return top 10 rated movies for a gold user', async () => {
22      // Arrange
23      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
24      Movie.find.mockResolvedValue(mockMovies);
25
26      // Act
27
28      // Assert
29    });
30  });
31
32  describe('Edge Cases', () => {
33    it('should return 480 if user tier is not found', async () => {
34      // Arrange
35      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
36      Movie.find.mockResolvedValue(mockMovies);
37
38      // Act
39
40      // Assert
41    });
42  });
43
44  describe('Error Cases', () => {
45    it('should return 400 if user tier is invalid', async () => {
46      // Arrange
47      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
48      Movie.find.mockResolvedValue(mockMovies);
49
50      // Act
51
52      // Assert
53    });
54  });
55
56  describe('Test Suites', () => {
57    it('should return top 10 rated movies for a platinum user (1 ms)', async () => {
58      // Arrange
59      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
60      Movie.find.mockResolvedValue(mockMovies);
61
62      // Act
63
64      // Assert
65    });
66  });
67
68  describe('Test Suites', () => {
69    it('should return top 10 rated movies for a silver user (1 ms)', async () => {
70      // Arrange
71      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
72      Movie.find.mockResolvedValue(mockMovies);
73
74      // Act
75
76      // Assert
77    });
78  });
79
80  describe('Test Suites', () => {
81    it('should return top 10 rated movies for a gold user (1 ms)', async () => {
82      // Arrange
83      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
84      Movie.find.mockResolvedValue(mockMovies);
85
86      // Act
87
88      // Assert
89    });
90  });
91
92  describe('Test Suites', () => {
93    it('should return top 10 rated movies for a bronze user (1 ms)', async () => {
94      // Arrange
95      const mockMovies = Array(10).fill({ title: 'Movie', rating: 5 });
96      Movie.find.mockResolvedValue(mockMovies);
97
98      // Act
99
100     // Assert
101   });
102 });

```

The bottom status bar shows the file path `PS D:\sw_final\Playbox-main (1)\Playbox-main\backend>` and various icons.

The image shows a VS Code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'backend', 'config', 'controllers', 'models', 'node\_modules', 'routes', 'utils', 'babel.config.js', 'env', 'fileMock.config.js', 'index.js', 'jest.config.js', 'package-lock.json', 'package.json', 'client', and 'ok'. The code editor displays a Jest test file named 'getRandomMovies.early.test.js'. The test file contains a series of Jest tests for a 'getRandomMovies' function. The tests include a 'describe' block for 'getRandomMovies' and a 'describe' block for 'Happy Paths'. The 'Happy Paths' block contains a 'it' test that checks if the function returns 10 random movies for a gold user. The test output is shown at the bottom of the editor, indicating that the test suite passed with 1 passed and 1 total test, and 6 passed and 6 total snapshots. The output also shows the estimated time for the test suite to run, which is 2.057 seconds.

```
getRandomMovies.early.test.js
1 // Unit tests for: getRandomMovies
2 import Movie from '../models/Movie.js';
3 import { getRandomMovies } from '../movieController';
4
5
6 jest.mock("../models/Movie.js");
7
8 describe('getRandomMovies() getRandomMovies method', () => {
9   let req, res;
10
11   beforeEach(() => {
12     req = {
13       user: {
14         tier: 'gold', // Default user tier for tests
15       },
16     };
17     res = {
18       status: jest.fn().mockReturnThis(),
19       json: jest.fn(),
20     };
21   });
22
23   describe('Happy Paths', () => {
24     it('should return 10 random movies for a gold user', async () => {
25       // Mock the aggregate function to return a list of movies
26       jest.mock('../models/Movie.js', () => {
27         return [
28           { title: 'Example Movie 1' },
29           { title: 'Example Movie 2' },
30           { title: 'Example Movie 3' },
31           { title: 'Example Movie 4' },
32           { title: 'Example Movie 5' },
33           { title: 'Example Movie 6' },
34           { title: 'Example Movie 7' },
35           { title: 'Example Movie 8' },
36           { title: 'Example Movie 9' },
37           { title: 'Example Movie 10' },
38         ];
39       });
40
41       const movies = await getRandomMovies(req, res);
42       expect(movies).toHaveLength(10);
43       expect(movies).toEqual(expect.arrayContaining([
44         { title: 'Example Movie 1' },
45         { title: 'Example Movie 2' },
46         { title: 'Example Movie 3' },
47         { title: 'Example Movie 4' },
48         { title: 'Example Movie 5' },
49         { title: 'Example Movie 6' },
50         { title: 'Example Movie 7' },
51         { title: 'Example Movie 8' },
52         { title: 'Example Movie 9' },
53         { title: 'Example Movie 10' },
54       ]));
55     });
56   });
57 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
✓ should return 10 random movies for a silver user (1 ms)
Edge Cases
✓ should return an error if user tier is not found
✓ should return an error if user tier is invalid (1 ms)
✓ should handle database errors gracefully

Test Suites: 1 passed, 1 total
Tests: 6 passed, 6 total
Snapshots: 0 total
Time: 2.057 s, estimated 8 s
Run all test suites matching /getRandomMovies.early.test.js/1
PS D:\sw_final\Playbox-main (1) Playbox-main\backend>
```

Ln 1, Col 1 Spaces: 2 UTF-8 CR LF JavaScript Go Live

# Specific Movie testing

```
1 // Unit tests for: getSpecificMovie
2 import Movie from "../../models/Movie.js";
3 import { getSpecificMovie } from "../movieController";
4
5 // Mock the Movie model
6 jest.mock("../../models/Movie.js");
7
8 describe('getSpecificMovie() getSpecificMovie method', () => {
9   let req, res;
10
11   beforeEach(() => {
12     req = {
13       params: {
14         id: 'someMovieId',
15       },
16     };
17
18     res = {
19       status: jest.fn().mockReturnThis(),
20       json: jest.fn(),
21     };
22   });
23
24   describe('Happy paths', () => {
25     it('should return the specific movie when found', async () => {
26       // ...
27     });
28   });
29 });
```

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 2.58 s, estimated 8 s  
Ran all test suites matching /getSpecificMovie.early.test.js/i.

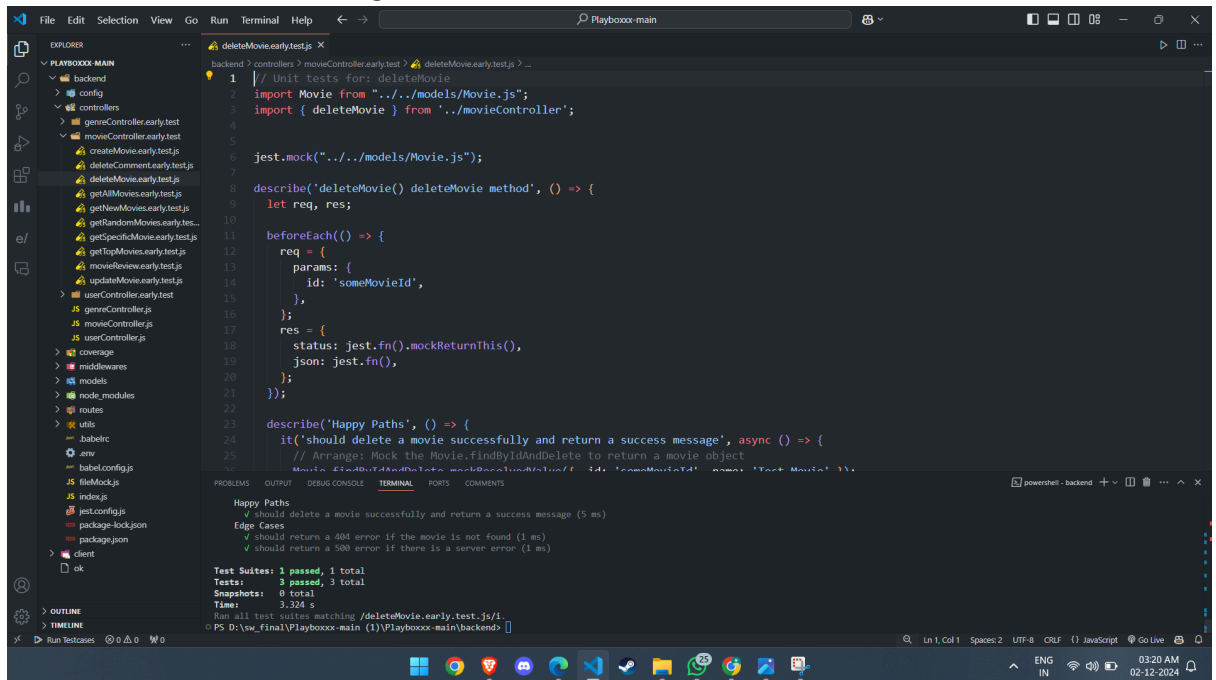
# All Movie testing

```
1 // Unit tests for: getAllMovies
2
3 import Movie from "../../models/Movie.js";
4 import { getAllMovies } from "../movieController";
5
6 jest.mock("../../models/Movie.js");
7
8 describe('getAllMovies() getAllMovies method', () => {
9   let req, res;
10
11   beforeEach(() => {
12     req = {
13       user: {
14         tier: 'gold',
15       },
16     };
17
18     res = {
19       status: jest.fn().mockReturnThis(),
20       json: jest.fn(),
21       send: jest.fn(),
22     };
23   });
24
25   // ...
26 });
```

Test Suites: 1 passed, 1 total  
Tests: 6 passed, 6 total  
Snapshots: 0 total  
Time: 2.12 s, estimated 8 s  
Ran all test suites matching /getAllMovies.early.test.js/i.



## Delete Movie testing



The screenshot shows a VS Code editor with a project named 'Playbox-main'. The Explorer panel on the left shows the file structure, with 'deleteMovie.early.test.js' selected. The main editor displays the following code:

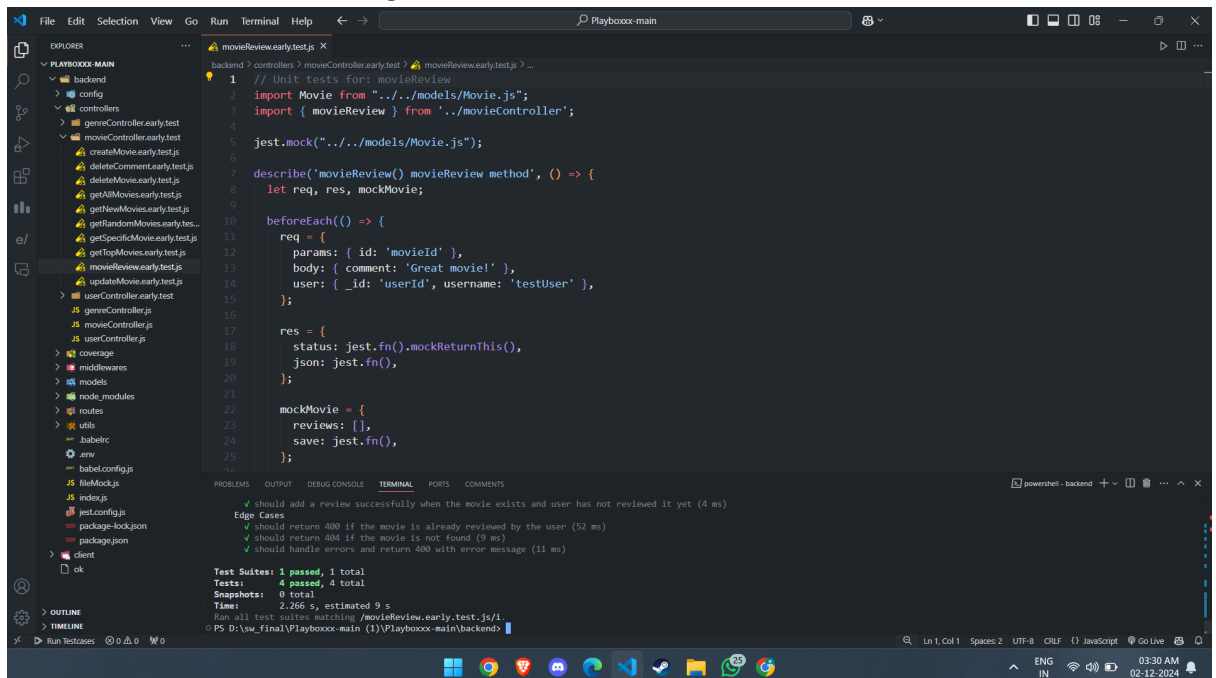
```
1 // Unit tests for: deleteMovie
2 import Movie from '../models/Movie.js';
3 import { deleteMovie } from '../movieController';
4
5
6 jest.mock('../models/Movie.js');
7
8 describe('deleteMovie() deleteMovie method', () => {
9   let req, res;
10
11   beforeEach(() => {
12     req = {
13       params: {
14         id: 'someMovieId',
15       },
16     };
17     res = {
18       status: jest.fn().mockReturnThis(),
19       json: jest.fn(),
20     };
21   });
22
23   describe('Happy Paths', () => {
24     it('should delete a movie successfully and return a success message', async () => {
25       // Arrange: Mock the Movie.findByIdAndDelete to return a movie object
26       Movie.findByIdAndDelete.mockResolvedValue({ id: 'someMovieId', name: 'Test Movie' });
```

The bottom panel shows the test results for 'deleteMovie.early.test.js/1'. The tests passed:

- Happy Paths
- ✓ should delete a movie successfully and return a success message (5 ms)
- Edge Cases
- ✓ should return a 404 error if the movie is not found (1 ms)
- ✓ should return a 500 error if there is a server error (1 ms)

Test Suites: 1 passed, 1 total  
Tests: 3 passed, 3 total  
Snapshots: 0 total  
Time: 3.324 s  
Ran all test suites matching /deleteMovie.early.test.js/1  
PS D:\sw\_final\Playbox-main (1)\Playbox-main\backends

## Movie Review testing



The screenshot shows a VS Code editor with a project named 'Playbox-main'. The Explorer panel on the left shows the file structure, with 'movieReview.early.test.js' selected. The main editor displays the following code:

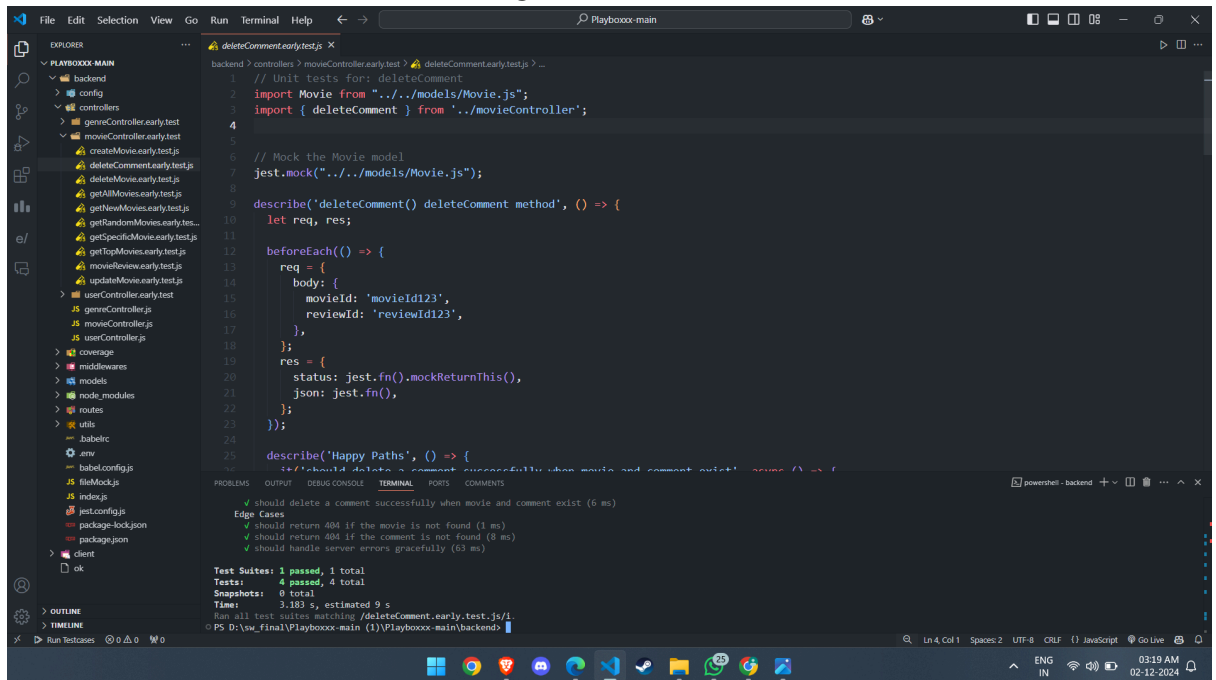
```
1 // Unit tests for: movieReview
2 import Movie from '../models/Movie.js';
3 import { movieReview } from '../movieController';
4
5 jest.mock('../models/Movie.js');
6
7 describe('movieReview() movieReview method', () => {
8   let req, res, mockMovie;
9
10   beforeEach(() => {
11     req = {
12       params: { id: 'movieId' },
13       body: { comment: 'Great movie!' },
14       user: { _id: 'userId', username: 'testUser' },
15     };
16     res = {
17       status: jest.fn().mockReturnThis(),
18       json: jest.fn(),
19     };
20     mockMovie = {
21       reviews: [],
22       save: jest.fn(),
23     };
24   });
25
26   it('should add a review successfully when the movie exists and user has not reviewed it yet (4 ms)
```

The bottom panel shows the test results for 'movieReview.early.test.js/1'. The tests passed:

- ✓ should add a review successfully when the movie exists and user has not reviewed it yet (4 ms)
- Edge Cases
- ✓ should return 400 if the movie is already reviewed by the user (52 ms)
- ✓ should return 404 if the movie is not found (9 ms)
- ✓ should handle errors and return 400 with error message (11 ms)

Test Suites: 1 passed, 1 total  
Tests: 4 passed, 4 total  
Snapshots: 0 total  
Time: 2.266 s, estimated 9 s  
Ran all test suites matching /movieReview.early.test.js/1  
PS D:\sw\_final\Playbox-main (1)\Playbox-main\backends

## Delete Movie Review testing



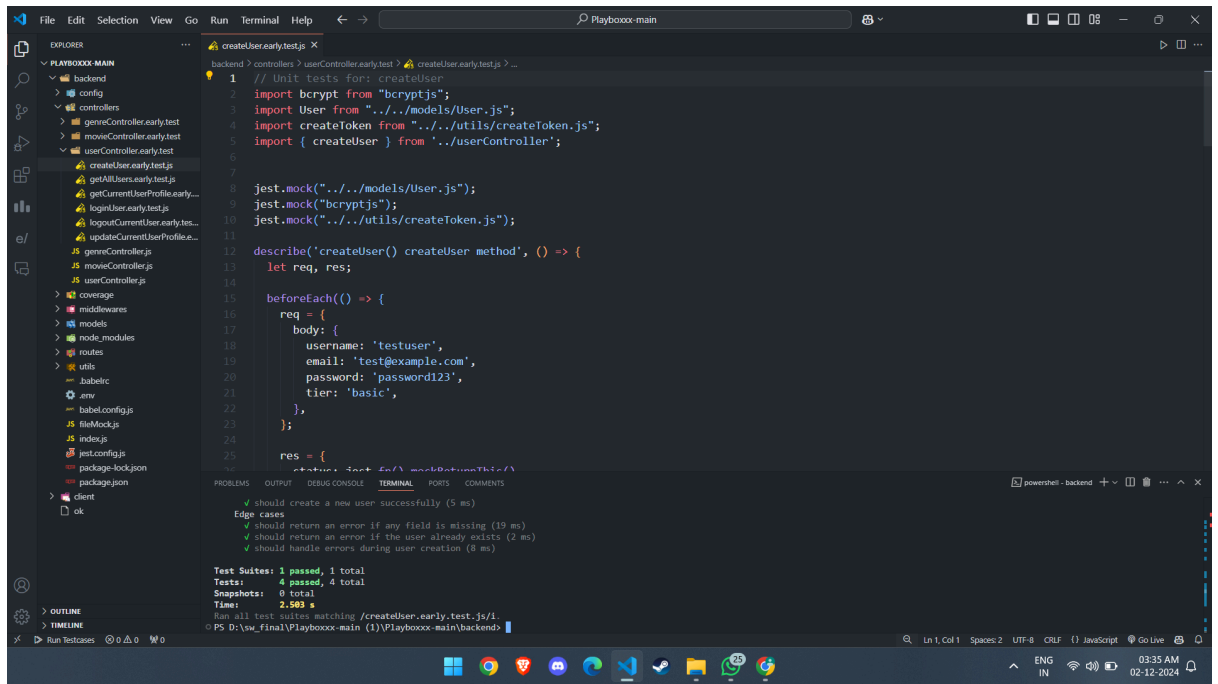
The screenshot shows a Visual Studio Code editor with a project named 'Playboxcc-main'. The Explorer sidebar on the left shows the file structure, including a 'controllers' directory with 'movieController.early.test.js' selected. The main editor displays the content of 'deleteComment.early.test.js', which contains Jest unit tests for the 'deleteComment' method. The tests include a 'beforeEach' block for mocking the Movie model, a 'describe' block for the 'deleteComment' method, and a 'describe' block for 'Happy Paths'. The terminal at the bottom shows the output of running the tests, indicating that all tests passed successfully.

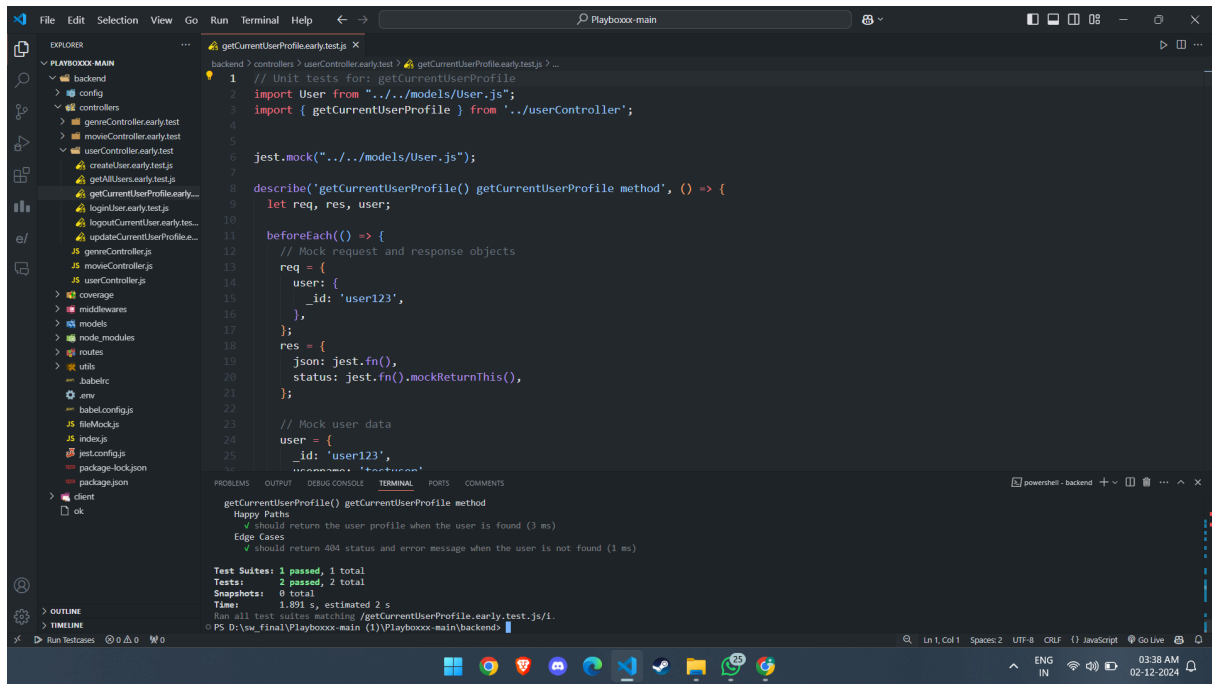
```
1 // Unit tests for: deleteComment
2 import Movie from "../../models/Movie.js";
3 import { deleteComment } from "../../movieController";
4
5
6 // Mock the Movie model
7 jest.mock("../../models/Movie.js");
8
9 describe('deleteComment() deleteComment method', () => {
10   let req, res;
11
12   beforeEach(() => {
13     req = {
14       body: {
15         movieId: 'movieId123',
16         reviewId: 'reviewId123',
17       },
18     };
19     res = {
20       status: jest.fn().mockReturnThis(),
21       json: jest.fn(),
22     };
23   });
24
25   describe('Happy Paths', () => {
26     it('should delete a comment successfully when movie and comment exist', () => {
27       // ...
28     });
29   });
30 });
```

Test Suites: 1 passed, 1 total  
Tests: 4 passed, 4 total  
Snapshots: 0 total  
Time: 3.183 s, estimated 9 s  
Ran all test suites matching /deleteComment.early.test.js/.

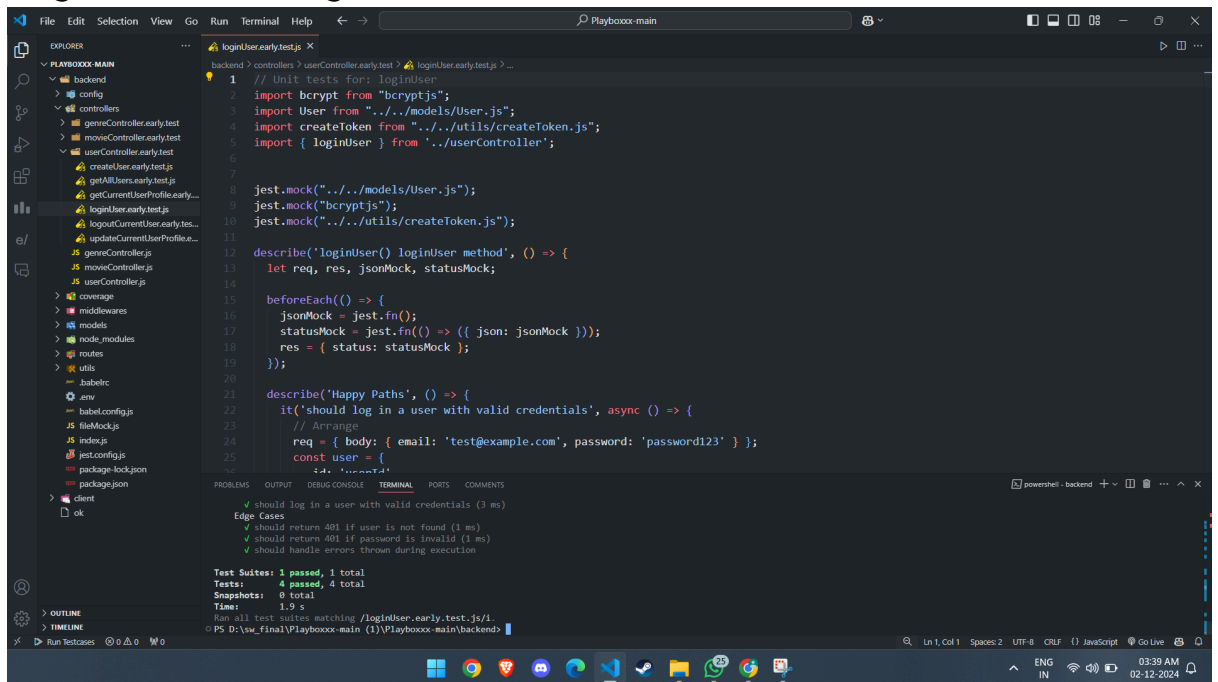
### 3. Testing of User Controller

#### Create User testing





## Login User testing



## ❖ **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.