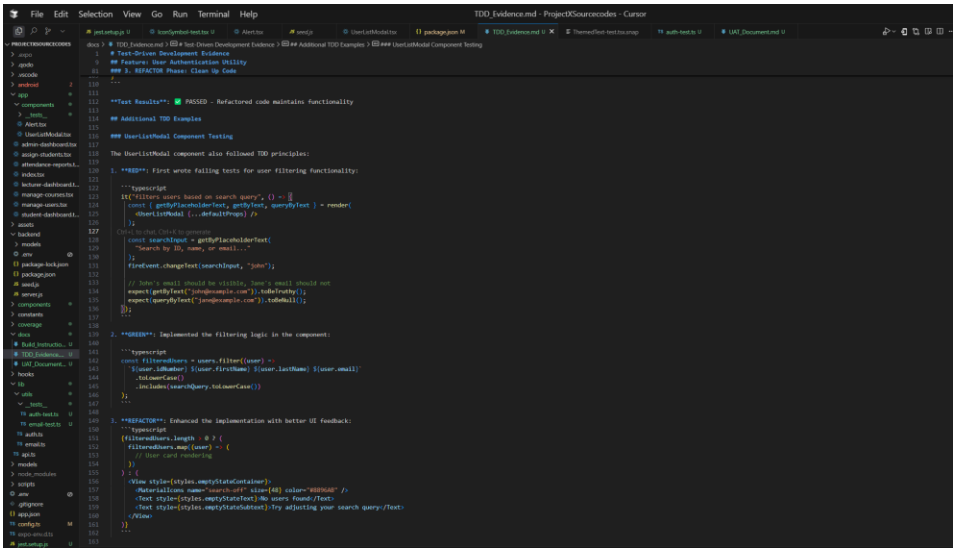
Test-Driven Development

TDD EVIDENCE



Source Code:

```
import { generateUsername, generatePassword, generateInitialPassword, authenticateUser } from '../auth';


// Mock fetch for authenticateUser
global.fetch = jest.fn();


describe('Auth Utils', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });


  describe('generateUsername', () => {
    it('generates username correctly from user information', () => {
      const firstName = 'John';
      const lastName = 'Doe';
      const idNumber = '2020-0001';
```

```
    const username = generateUsername(firstName, lastName, idNumber);

    // Should take first 3 letters of first name, first 3 letters of last name, and last 4 digits of ID
    expect(username).toBe('johdoe0001');
  });

  it('handles names with spaces correctly', () => {
    const firstName = 'John James';
    const lastName = 'Doe Smith';
    const idNumber = '2020-0001';

    const username = generateUsername(firstName, lastName, idNumber);

    // Should remove spaces and take first 3 letters of each name
    expect(username).toBe('johdoe0001');
  });

  it('handles short names correctly', () => {
    const firstName = 'Jo';
    const lastName = 'Do';
    const idNumber = '2020-0001';

    const username = generateUsername(firstName, lastName, idNumber);

    // Should take available letters even if less than 3
    expect(username).toBe('jodo0001');
  });
});

describe('generatePassword', () => {
  it('generates a password of correct length', () => {
```

```
    const password = generatePassword();

    // Should be 8 characters (4 bytes converted to hex)
    expect(password.length).toBe(8);
  });

  it('generates different passwords on each call', () => {
    const password1 = generatePassword();
    const password2 = generatePassword();

    // Passwords should be different
    expect(password1).not.toBe(password2);
  });
});

describe('generateInitialPassword', () => {
  it('generates a password based on user information', () => {
    const firstName = 'John';
    const lastName = 'Doe';
    const idNumber = '2020-0001';

    const password = generateInitialPassword(firstName, lastName, idNumber);

    // Should start with first letter of first name, first letter of last name, and last 4 digits of ID
    expect(password).toMatch(/^jd0001@\d{1,3}$/);
  });

  it('handles names with spaces correctly', () => {
    const firstName = 'John James';
    const lastName = 'Doe Smith';
    const idNumber = '2020-0001';
```

```
    const password = generateInitialPassword(firstName, lastName, idNumber);

    // Should use first letter of first name and first letter of last name
    expect(password).toMatch(/^jd0001@\d{1,3}$/);
  });
});

describe('authenticateUser', () => {
  it('authenticates user successfully', async () => {
    const mockResponse = {
      token: 'test-token',
      user: {
        id: '1',
        username: 'testuser',
        role: 'student',
      },
    };

    (global.fetch as jest.Mock).mockResolvedValueOnce({
      ok: true,
      json: jest.fn().mockResolvedValueOnce(mockResponse),
    });

    const result = await authenticateUser('testuser', 'password123');

    expect(result).toEqual(mockResponse);
    expect(global.fetch).toHaveBeenCalledWith(
      'http://192.168.0.247:3000/api/auth/login',
      {
        method: 'POST',
```

```
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username: 'testuser', password: 'password123' }),
      }
    );
  });

  it('handles authentication failure', async () => {
    const errorMessage = 'Invalid credentials';

    (global.fetch as jest.Mock).mockResolvedValueOnce({
      ok: false,
      status: 401,
      json: jest.fn().mockResolvedValueOnce({ message: errorMessage }),
    });

    await expect(authenticateUser('testuser', 'wrongpassword')).rejects.toThrow(
      `Authentication failed: ${errorMessage}`
    );
  });

  it('handles network errors', async () => {
    (global.fetch as jest.Mock).mockRejectedValueOnce(new Error('Network error'));

    await expect(authenticateUser('testuser', 'password123')).rejects.toThrow(
      'Authentication failed: Network error'
    );
  });
 });
});
```