IT314: Software Engineering
Group - 4 (Real Estate Management System)

Unit Testing

We have used **Jest** for writing test cases, which is a testing framework designed for JavaScript applications.

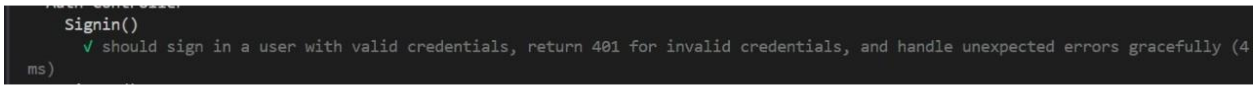
Testing Framework : `"jest": "^29.7.0"`
Assertion library : `"babel-jest": "^29.7.0"`
Other: `"sinon": "^19.0.2"`

Auth Controller:

a) Signin():

```
describe('Signin()', () => {
        let req, res, next;

        beforeEach(() => {
            req = {
                body: {
                    email: 'test@example.com',
                    password: 'password123'
                }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn(),
                cookie: jest.fn().mockReturnThis()
            };
            next = jest.fn();
        });

        // Combined test suite for happy path and edge cases
        it('should sign in a user with valid credentials, return 401 for
invalid credentials, and handle unexpected errors gracefully', async () =>
{
            // Happy Path: Valid credentials
            const mockUser = {
                _id: 'userId123',
                email: 'test@example.com',
                password: 'hashedPassword',
                _doc: { email: 'test@example.com', username: 'testuser' }
            };
            User.findOne.mockResolvedValue(mockUser);
            bcryptjs.compareSync.mockReturnValue(true);
            jwt.sign.mockReturnValue('mockToken');

            await signin(req, res, next);
            expect(User.findOne).toHaveBeenCalledWith({ email:
'test@example.com' });

expect(bcryptjs.compareSync).toHaveBeenCalledWith('password123',
'hashedPassword');
            expect(jwt.sign).toHaveBeenCalledWith({ id: 'userId123' },
process.env.JWT_SECRET);
            expect(res.cookie).toHaveBeenCalledWith('access_token',
'mockToken', { httpOnly: true });
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith({ email:
'test@example.com', username: 'testuser' });

            // Edge Case: User does not exist
            User.findOne.mockResolvedValue(null);
            await signin(req, res, next);
            expect(User.findOne).toHaveBeenCalledWith({ email:
'test@example.com' });
            expect(next).toHaveBeenCalledWith(errorHandler(401, 'Invalid
E-mail or Password'));

            // Edge Case: Incorrect password
            User.findOne.mockResolvedValue(mockUser);
            bcryptjs.compareSync.mockReturnValue(false);
            await signin(req, res, next);

expect(bcryptjs.compareSync).toHaveBeenCalledWith('password123',
'hashedPassword');
            expect(next).toHaveBeenCalledWith(errorHandler(401, 'Invalid
E-mail or Password!'));

            // Edge Case: Unexpected error
            const error = new Error('Unexpected Error');
            User.findOne.mockRejectedValue(error);
            await signin(req, res, next);
            expect(next).toHaveBeenCalledWith(error);
        });
    });
```

Results:

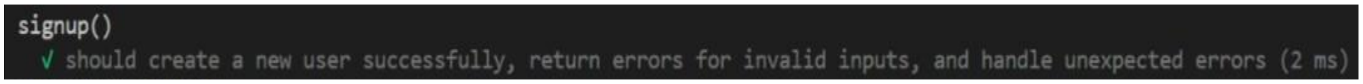


```
    Signin()
      √ should sign in a user with valid credentials, return 401 for invalid credentials, and handle unexpected errors gracefully (4
 ms)
```

b) SignUp():

```
describe('signup()', () => {
    let req, res, next;

    beforeEach(() => {
        req = {
            body: {
                username: 'testuser',
                email: 'test@example.com',
                password: 'Password123!'
            }
        };
        res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };
        next = jest.fn();
    });

    it('should create a new user successfully, return errors for invalid
inputs, and handle unexpected errors', async () => {
        // Happy Path: Successful user creation
        User.findOne.mockResolvedValue(null);  // No user exists with the
same username or email
        validateEmail.mockReturnValue(true);  // Email is valid
        validatePassword.mockReturnValue(null);  // Password is valid
        bcryptjs.hashSync.mockReturnValue('hashedPassword');  // Mock
bcrypt hash
        User.prototype.save = jest.fn().mockResolvedValue();  // Simulate
successful user save

        await signup(req, res, next);

        // Assert successful user creation
        expect(User.findOne).toHaveBeenCalledWith({ username: 'testuser'
});
        expect(validateEmail).toHaveBeenCalledWith('test@example.com');
        expect(validatePassword).toHaveBeenCalledWith('Password123!');
        expect(bcryptjs.hashSync).toHaveBeenCalledWith('Password123!',
10);
        expect(User.prototype.save).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(201);
        expect(res.json).toHaveBeenCalledWith('User Created
Successfully!');

        // Edge Case: Username already exists
        User.findOne.mockResolvedValueOnce({ username: 'testuser' });  // Mock user with the same username
        await signup(req, res, next);
        expect(next).toHaveBeenCalledWith(errorHandler(409, 'Username already Exists!'));

        // Edge Case: Invalid email format
        User.findOne.mockResolvedValueOnce(null);  // No user exists
        validateEmail.mockReturnValueOnce(false);  // Invalid email
        await signup(req, res, next);
        expect(next).toHaveBeenCalledWith(errorHandler(400, 'Invalid Email Format'));

        // Edge Case: Email already exists
        User.findOne
            .mockResolvedValueOnce(null)  // No user with the same username
            .mockResolvedValueOnce({ email: 'test@example.com' });  // User with the same email exists
        validateEmail.mockReturnValue(true);  // Email format is valid
        await signup(req, res, next);
        expect(next).toHaveBeenCalledWith(errorHandler(409, 'Email already Exists!'));

        // Edge Case: Weak password
        validatePassword.mockReturnValueOnce('Password is too weak');  // Password is too weak
        await signup(req, res, next);
        expect(next).toHaveBeenCalledWith(errorHandler(400, 'Password is too weak'));

        // Edge Case: Error during user creation (Database error)
        User.prototype.save = jest.fn().mockRejectedValueOnce(new Error('Database error'));  // Simulate DB error
        await signup(req, res, next);
        expect(next).toHaveBeenCalledWith(new Error('Database error'));
    });
});
```

Results:

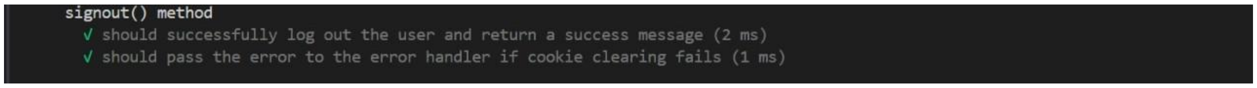


```
signup()
  √ should create a new user successfully, return errors for invalid inputs, and handle unexpected errors (2 ms)
```

c) SignOut

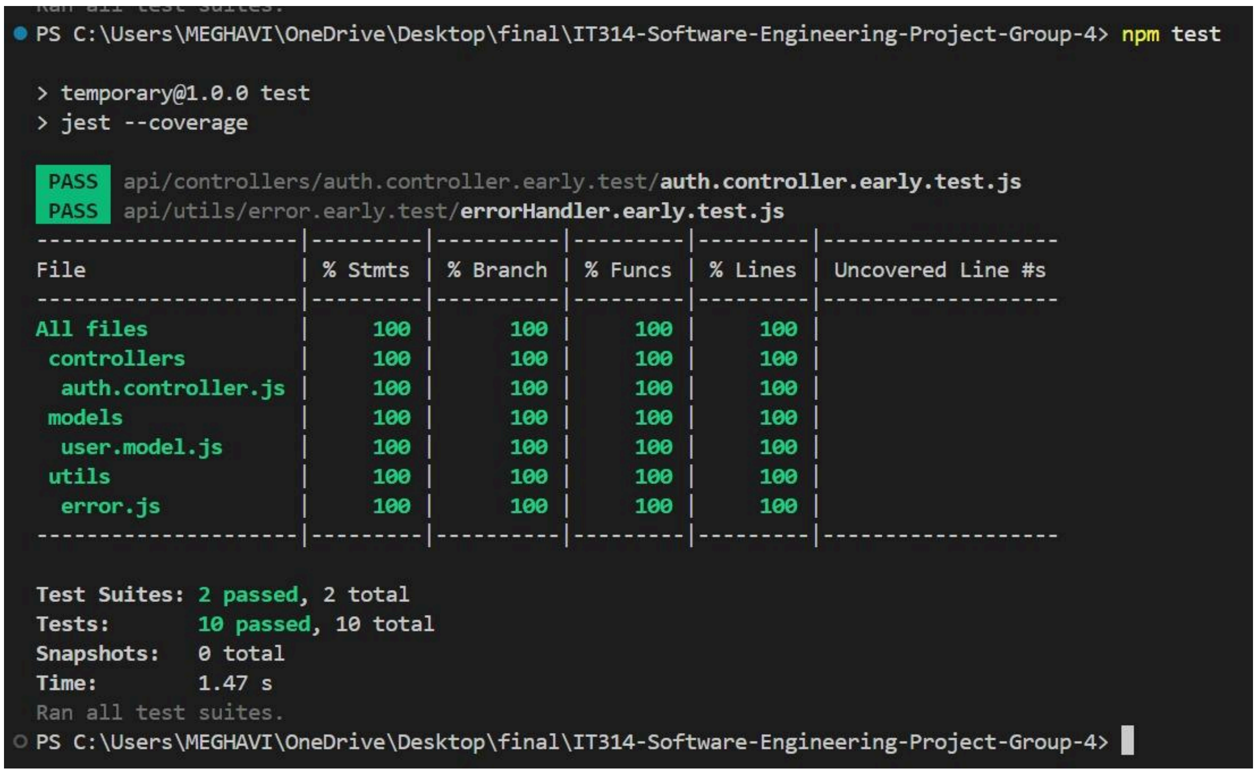
```
describe('signout() method', () => {
    it('should successfully log out the user and return a success
message', async () => {
        // Arrange: Create mock request and response objects
        const req = httpMocks.createRequest();
        const res = httpMocks.createResponse();
        const next = jest.fn();

        // Mock clearCookie to not throw any error (success scenario)
        res.clearCookie = jest.fn();

        // Mock response methods
        res.status = jest.fn().mockReturnValue(res);  // Mock status
method
        res.json = jest.fn();  // Mock json method

        // Act: Call the signout function
        await signout(req, res, next);

        // Assert: Check that res.status() and res.json() were called
correctly
        expect(res.status).toHaveBeenCalledWith(200);  // Expect 200
status code
        expect(res.json).toHaveBeenCalledWith('User has been Logged Out
Successfully!');  // Expect success message
    });
    it('should pass the error to the error handler if cookie clearing fails', async () => {
        // Arrange: Create mock request and response objects
        const req = httpMocks.createRequest();
        const res = httpMocks.createResponse();
        const next = jest.fn();

        // Mock the clearCookie method to throw an error
        res.clearCookie = jest.fn(() => {
            throw new Error('Failed to clear cookie');
        });

        // Act: Call the signout function
        await signout(req, res, next);

        // Assert: Ensure next is called with an error
        expect(next).toHaveBeenCalledWith(expect.objectContaining({
            message: 'Failed to clear cookie',
            statusCode: 500
        }));
    });
});
```

Results:



```
    signout() method
      √ should successfully log out the user and return a success message (2 ms)
      √ should pass the error to the error handler if cookie clearing fails (1 ms)
```

Code Coverage:



```
PS C:\Users\MEGHAVI\OneDrive\Desktop\final\IT314-Software-Engineering-Project-Group-4> npm test

> temporary@1.0.0 test
> jest --coverage

 PASS  api/controllers/auth.controller.early.test/auth.controller.early.test.js
 PASS  api/utils/error.early.test/errorHandler.early.test.js
---------------------|---------|----------|---------|---------|-------------------
File                 | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------|---------|----------|---------|---------|-------------------
All files            |     100 |      100 |     100 |     100 |
 controllers         |     100 |      100 |     100 |     100 |
  auth.controller.js |     100 |      100 |     100 |     100 |
 models              |     100 |      100 |     100 |     100 |
  user.model.js      |     100 |      100 |     100 |     100 |
 utils               |     100 |      100 |     100 |     100 |
  error.js           |     100 |      100 |     100 |     100 |
---------------------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        1.47 s
Ran all test suites.
PS C:\Users\MEGHAVI\OneDrive\Desktop\final\IT314-Software-Engineering-Project-Group-4>
```