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.

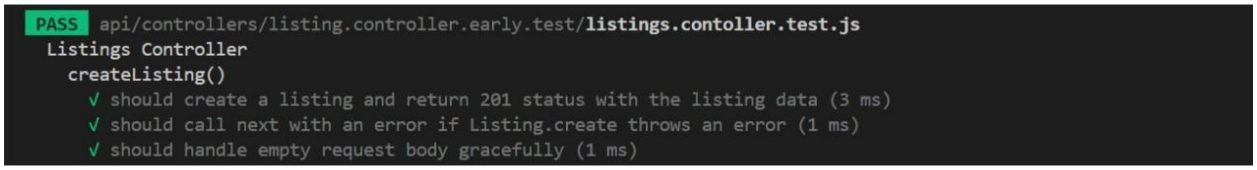
Testing Framework : `"jest": "^29.7.0"`
Assertion library : `"babel-jest": "^29.7.0"`
Other: `"sinon": "^19.0.2"`

Listing Controller:

a) CreateListings():

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

        beforeEach(() => {
            req = {
                body: {
                    title: 'Test Listing',
                    description: 'A test listing description',
                    price: 100
                }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should create a listing and return 201 status with the listing
data', async () => {
            // Arrange
            const mockListing = { id: '1', ...req.body };
            Listing.create.mockResolvedValue(mockListing);

            // Act
            await createListing(req, res, next);

            // Assert
            expect(Listing.create).toHaveBeenCalledWith(req.body);
            expect(res.status).toHaveBeenCalledWith(201);
            expect(res.json).toHaveBeenCalledWith(mockListing);
        });

        it('should call next with an error if Listing.create throws an
error', async () => {
            // Arrange
            const error = new Error('Database error');
            Listing.create.mockRejectedValue(error);

            // Act
            await createListing(req, res, next);

            // Assert
            expect(next).toHaveBeenCalledWith(error);
        });

        it('should handle empty request body gracefully', async () => {
            // Arrange
            req.body = {}; // Empty body
            const mockListing = { id: '1', ...req.body };
            Listing.create.mockResolvedValue(mockListing);

            // Act
            await createListing(req, res, next);

            // Assert
            expect(Listing.create).toHaveBeenCalledWith(req.body);
            expect(res.status).toHaveBeenCalledWith(201);
            expect(res.json).toHaveBeenCalledWith(mockListing);
        });
    });
```

Result:

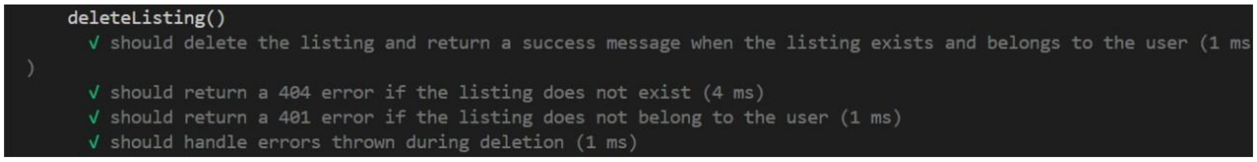


```
 PASS  api/controllers/listing.controller.early.test/listings.contoller.test.js
  Listings Controller
    createListing()
      √ should create a listing and return 201 status with the listing data (3 ms)
      √ should call next with an error if Listing.create throws an error (1 ms)
      √ should handle empty request body gracefully (1 ms)
```

b) DeleteListings()

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

        beforeEach(() => {
            req = {
                params: { id: '123' },
                user: { id: 'user123' }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should delete the listing and return a success message when
the listing exists and belongs to the user', async () => {
            // Arrange
            const mockListing = { _id: '123', userRef: 'user123' };
            Listing.findById.mockResolvedValue(mockListing);
            Listing.findByIdAndDelete.mockResolvedValue(mockListing);

            // Act
            await deleteListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(Listing.findByIdAndDelete).toHaveBeenCalledWith('123');
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith('Listing has been
deleted!');
        });

        it('should return a 404 error if the listing does not exist',
async () => {
            // Arrange
            Listing.findById.mockResolvedValue(null);
            errorHandler.mockReturnValue(new Error('Listing Not Found!'));

            // Act
            await deleteListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(new Error('Listing Not
Found!'));
        });

        it('should return a 401 error if the listing does not belong to
the user', async () => {
            // Arrange
            const mockListing = { _id: '123', userRef: 'otherUser' };
            Listing.findById.mockResolvedValue(mockListing);
            errorHandler.mockReturnValue(new Error('You can only delete
your own listings!'));

            // Act
            await deleteListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(new Error('You can only
delete your own listings!'));
        });

        it('should handle errors thrown during deletion', async () => {
            // Arrange
            const mockListing = { _id: '123', userRef: 'user123' };
            Listing.findById.mockResolvedValue(mockListing);
            const deletionError = new Error('Deletion failed');
            Listing.findByIdAndDelete.mockRejectedValue(deletionError);

            // Act
            await deleteListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(Listing.findByIdAndDelete).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(deletionError);
        });
    });
```

Result:

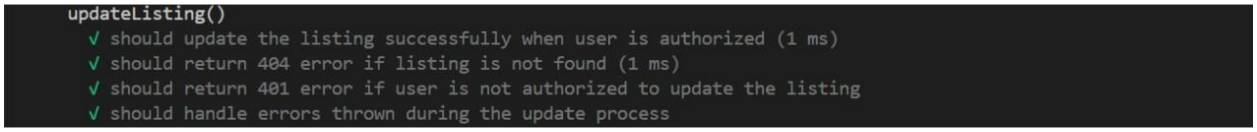


```
    deleteListing()
      √ should delete the listing and return a success message when the listing exists and belongs to the user (1 ms
)
      √ should return a 404 error if the listing does not exist (4 ms)
      √ should return a 401 error if the listing does not belong to the user (1 ms)
      √ should handle errors thrown during deletion (1 ms)
```

c) UpdateListings():

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

        beforeEach(() => {
            req = {
                params: { id: '123' },
                body: { title: 'Updated Title' },
                user: { id: 'user123' }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should update the listing successfully when user is
authorized', async () => {
            // Arrange
            const listing = { _id: '123', userRef: 'user123' };
            const updatedListing = { _id: '123', title: 'Updated Title',
userRef: 'user123' };
            Listing.findById.mockResolvedValue(listing);
            Listing.findByIdAndUpdate.mockResolvedValue(updatedListing);

            // Act
            await updateListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(Listing.findByIdAndUpdate).toHaveBeenCalledWith('123',
req.body, { new: true });
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith(updatedListing);
        });

        it('should return 404 error if listing is not found', async () =>
{
            // Arrange
            Listing.findById.mockResolvedValue(null);
            const error = { status: 404, message: 'Listing Not Found!' };
            errorHandler.mockReturnValue(error);

            // Act
            await updateListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(error);
        });

        it('should return 401 error if user is not authorized to update
the listing', async () => {
            // Arrange
            const listing = { _id: '123', userRef: 'anotherUser' };
            Listing.findById.mockResolvedValue(listing);
            const error = { status: 401, message: 'You can only update
your own listings!' };
            errorHandler.mockReturnValue(error);

            // Act
            await updateListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(error);
        });

        it('should handle errors thrown during the update process', async
() => {
            // Arrange
            const listing = { _id: '123', userRef: 'user123' };
            Listing.findById.mockResolvedValue(listing);
            const error = new Error('Database error');
            Listing.findByIdAndUpdate.mockRejectedValue(error);

            // Act
            await updateListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(Listing.findByIdAndUpdate).toHaveBeenCalledWith('123',
req.body, { new: true });
            expect(next).toHaveBeenCalledWith(error);
        });
    });
```

Result:

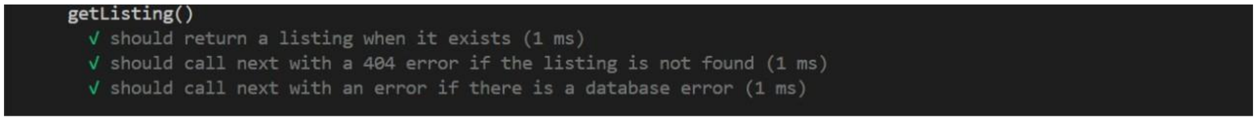


```
    updateListing()
      √ should update the listing successfully when user is authorized (1 ms)
      √ should return 404 error if listing is not found (1 ms)
      √ should return 401 error if user is not authorized to update the listing
      √ should handle errors thrown during the update process
```

d) GetListings()

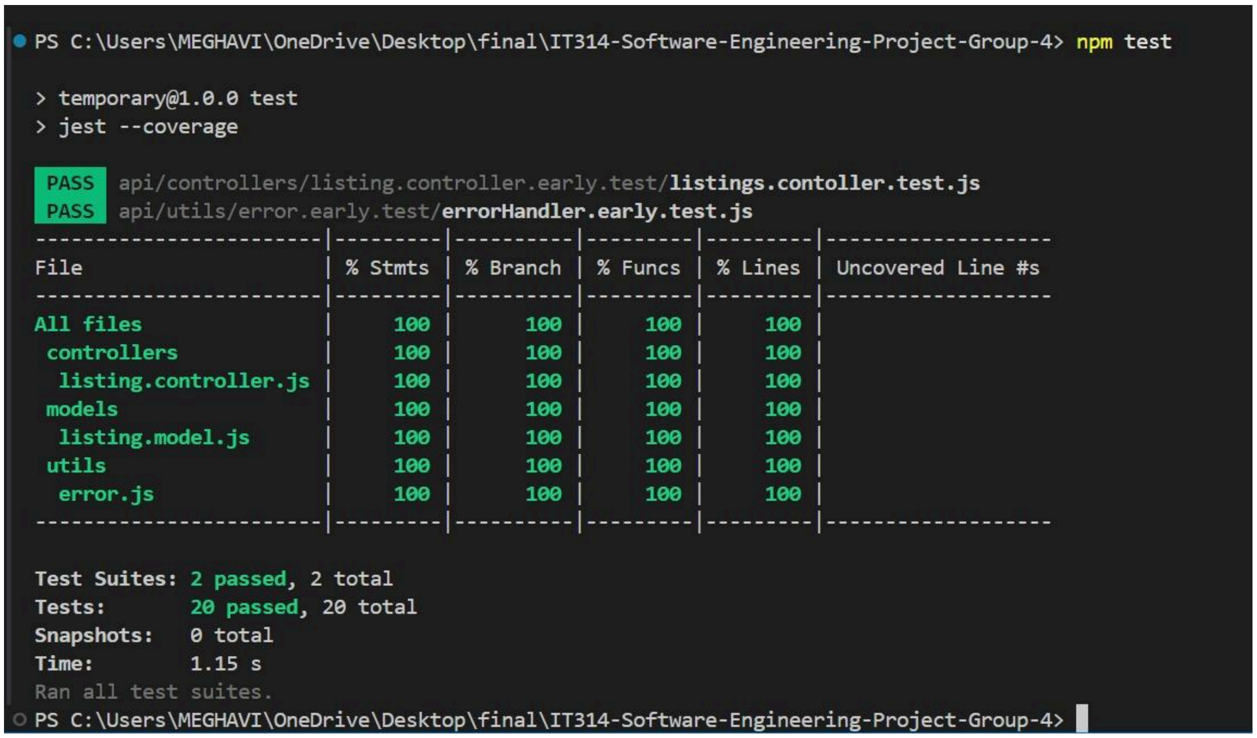
```
describe('getListing()', () => {
        let req, res, next;

        beforeEach(() => {
            req = {
                params: { id: '123' }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should return a listing when it exists', async () => {
            // Arrange
            const mockListing = { id: '123', title: 'Test Listing' };
            Listing.findById.mockResolvedValue(mockListing);

            // Act
            await getListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith(mockListing);
        });

        it('should call next with a 404 error if the listing is not
found', async () => {
            // Arrange
            Listing.findById.mockResolvedValue(null);
            const error = new Error('Listing Not Found!');
            errorHandler.mockReturnValue(error);

            // Act
            await getListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(error);
        });

        it('should call next with an error if there is a database error',
async () => {
            // Arrange
            const dbError = new Error('Database error');
            Listing.findById.mockRejectedValue(dbError);

            // Act
            await getListing(req, res, next);

            // Assert
            expect(Listing.findById).toHaveBeenCalledWith('123');
            expect(next).toHaveBeenCalledWith(dbError);
        });
    });
```

Result:



```
    getListing()
      √ should return a listing when it exists (1 ms)
      √ should call next with a 404 error if the listing is not found (1 ms)
      √ should call next with an error if there is a database error (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/listing.controller.early.test/listings.contoller.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 |
  listing.controller.js |     100 |      100 |     100 |     100 |
 models                 |     100 |      100 |     100 |     100 |
  listing.model.js      |     100 |      100 |     100 |     100 |
 utils                  |     100 |      100 |     100 |     100 |
  error.js              |     100 |      100 |     100 |     100 |
------------------------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       20 passed, 20 total
Snapshots:   0 total
Time:        1.15 s
Ran all test suites.
PS C:\Users\MEGHAVI\OneDrive\Desktop\final\IT314-Software-Engineering-Project-Group-4>
```