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"`

Transaction Controller:

(a)getTransaction();

```
describe('getTransaction()', () => {
       it('should return a transaction when found', async () => {
           // Arrange
           const mockTransaction = { id: 'transaction123', userId:
'user123', listingId: 'listing123' };
           Transaction.findOne.mockResolvedValue(mockTransaction);

           // Act
           await getTransaction(req, res, next);
```

```
            // Assert
            expect(Transaction.findOne).toHaveBeenCalledWith({
userId: 'user123', listingId: 'listing123' });
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith(mockTransaction);
        });
        it('should return 204 when no transaction is found', async
() => {
            // Arrange
            Transaction.findOne.mockResolvedValue(null);

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

            // Assert
            expect(Transaction.findOne).toHaveBeenCalledWith({
userId: 'user123', listingId: 'listing123' });
            expect(res.status).toHaveBeenCalledWith(204);
            expect(res.json).toHaveBeenCalledWith('No Transaction
Found!');
        });

        it('should call next with an error if an exception is
thrown', async () => {
            // Arrange
            const error = new Error('Database error');
            Transaction.findOne.mockRejectedValue(error);

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

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


    });
```

Result:

```
 PASS  api/controllers/test/transaction.test.js
  transactionController
    getTransaction()
      √ should return a transaction when found (3 ms)
      √ should return 204 when no transaction is found
      √ should call next with an error if an exception is thrown (1 ms)
```

(b)getTransactions();

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

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

        it('should return a list of transactions for the user',
async () => {
            // Mock data
            const sentTransactions = [
                { _doc: { id: '1', createdAt: '2023-10-01T10:00:00Z'
} },
                { _doc: { id: '2', createdAt: '2023-10-02T10:00:00Z'
} }
            ];
            const receivedTransactions = [
                { _doc: { id: '3', createdAt: '2023-10-03T10:00:00Z'
} }
            ];
```

```
            // Mock the Transaction model methods
            Transaction.find.mockImplementation((query) => {
                if (query.userId) return
Promise.resolve(sentTransactions);
                if (query.sellerId) return
Promise.resolve(receivedTransactions);
            });

            await getTransactions(req, res, next);

            expect(Transaction.find).toHaveBeenCalledTimes(2);
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith([
                { id: '3', createdAt: '2023-10-03T10:00:00Z', type:
'received' },
                { id: '2', createdAt: '2023-10-02T10:00:00Z', type:
'sent' },
                { id: '1', createdAt: '2023-10-01T10:00:00Z', type:
'sent' }
            ]);
        });
        it('should return an empty array if no transactions are
found', async () => {
            // Mock the Transaction model methods to return empty
arrays
            Transaction.find.mockResolvedValue([]);

            await getTransactions(req, res, next);

            expect(Transaction.find).toHaveBeenCalledTimes(4);
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith([]);
        });
        it('should call next with an error if user ID does not
match', async () => {
            req.params.id = 'differentUserId';
            req.user.id ='someOtherUserId';

            await getTransactions(req, res, next);
```

```
expect(next).toHaveBeenCalledWith(expect.objectContaining({
                status: 401,
                message: 'You can only view your own transactions!'
            }));
        });

        it('should call next with an error if there is a database
error', async () => {
            const error = new Error('Database error');
            Transaction.find.mockRejectedValue(error);

            await getTransactions(req, res, next);

            expect(next).toHaveBeenCalledWith(error);
        });
    });
```

Result:

```
getTransactions
  √ should return a list of transactions for the user (1 ms)
  √ should return an empty array if no transactions are found (1 ms)
  √ should call next with an error if user ID does not match (1 ms)
  √ should call next with an error if there is a database error (1 ms)
```

(c) createTransactions();

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

    beforeEach(() => {
        req = {
            body: {
                // Sample transaction data
                userId: 'user123',
                listingId: 'listing123',
                amount: 100,
                // Add other necessary fields
            }
```

```
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should create a transaction successfully and return a 201
status', async () => {
            // Arrange
            Transaction.create.mockResolvedValueOnce({});

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

            // Assert
            expect(Transaction.create).toHaveBeenCalledWith(req.body);
            expect(res.status).toHaveBeenCalledWith(201);
            expect(res.json).toHaveBeenCalledWith('Transaction Completed
Successfully');
        });
                it('should handle errors during transaction creation and
call next with error', async () => {
            // Arrange
            const error = new Error('Database error');
            Transaction.create.mockRejectedValueOnce(error);

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

            // Assert
            expect(Transaction.create).toHaveBeenCalledWith(req.body);
            expect(next).toHaveBeenCalledWith(error);
        });

        it('should handle missing transaction data gracefully', async ()
=> {
            // Arrange
            req.body = {}; // Simulate missing data
```

```
            const error = new Error('Validation error');
            Transaction.create.mockRejectedValueOnce(error);

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

            // Assert
            expect(Transaction.create).toHaveBeenCalledWith(req.body);
            expect(next).toHaveBeenCalledWith(error);
        });
        });
    });
```

Result:

```
createTransactions()
  ✓ should create a transaction successfully and return a 201 status (1 ms)
  ✓ should handle errors during transaction creation and call next with error (1 ms)
  ✓ should handle missing transaction data gracefully (1 ms)
```

Code Coverage:

```
PASS api/controllers/test/transaction.test.js
----------------------------|---------|----------|---------|---------|-------------------
File                        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------|---------|----------|---------|---------|-------------------
All files                   |     100 |      100 |     100 |     100 |
 controllers                |     100 |      100 |     100 |     100 |
  transaction.controller.js |     100 |      100 |     100 |     100 |
 models                     |     100 |      100 |     100 |     100 |
  transaction.model.js      |     100 |      100 |     100 |     100 |
 utils                      |     100 |      100 |     100 |     100 |
  error.js                  |     100 |      100 |     100 |     100 |
  validation.js             |     100 |      100 |     100 |     100 |
----------------------------|---------|----------|---------|---------|-------------------
```