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"`

Property Controller:

a) Property Search():

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

  beforeEach(() => {
    req = {
      query: {
        searchQuery: 'Ahmedabad',
      },
    };
    res = {
```

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

  afterEach(() => {
    sinon.restore();
  });

  it('should return properties if searchQuery is provided', async () => {
    const mockProperties = [{ name: 'Ahmedabad' }];
    sinon.stub(Listing, 'find').resolves(mockProperties);

    req.query = { searchQuery: 'Ahmedabad' };

    await search(req, res, next);

    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith(mockProperties);
  });

  it('should return empty array if no properties match searchQuery', async
() => {
    const mockProperties = [];
    sinon.stub(Listing, 'find').resolves(mockProperties);

    req.query = { searchQuery: 'NonexistentCity' };

    await search(req, res, next);

    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith(mockProperties);
  });

  it('should return a message if no searchQuery is provided', async () =>
{
    req.query = {};

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

```
    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith({ message: 'Could not find' });
  });

  it('should return an error if database query fails', async () => {
    sinon.stub(Listing, 'find').rejects(new Error('Database error'));

    req.query = { searchQuery: 'Ahmedabad' };

    await search(req, res, next);

    expect(res.status).toHaveBeenCalledWith(302);
    expect(res.json).toHaveBeenCalledWith({ message: 'Error fetching
properties' });
  });

  it('should handle invalid searchQuery gracefully', async () => {
    sinon.stub(Listing, 'find').resolves([]);

    req.query = { searchQuery: 'Invalid*Query' };

    await search(req, res, next);

    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith([]);
  });

  it('should pass error to next middleware in case of failure', async ()
=> {
    const error = new Error('error');
    sinon.stub(Listing, 'find').rejects(error);

    req.query = { searchQuery: 'Ahmedabad' };

    await search(req, res, next);

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

```
  it('should use regex in query for searchQuery', async () => {
    const mockProperties = [{ name: 'Ahmedabad' }];
    const findStub = sinon.stub(Listing, 'find').resolves(mockProperties);

    req.query = { searchQuery: 'Ahmedabad' };

    await search(req, res, next);

    const queryPassedToFind = findStub.firstCall.args[0];
    expect(queryPassedToFind).toHaveProperty('$or');
    expect(queryPassedToFind.$or[0]).toHaveProperty('name', { $regex:
'Ahmedabad', $options: 'i' });
  });

  it('should return properties with correct structure when searchQuery
matches', async () => {
    const mockProperties = [
      { name: 'Ahmedabad', city: 'Gujarat', pinCode: '380001' },
    ];
    sinon.stub(Listing, 'find').resolves(mockProperties);

    req.query = { searchQuery: 'Ahmedabad' };

    await search(req, res, next);

    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith(mockProperties);
  });

  it('should return properties case-insensitively', async () => {
    const mockProperties = [{ name: 'Ahmedabad' }];
    sinon.stub(Listing, 'find').resolves(mockProperties);

    req.query = { searchQuery: 'ahmedabad' };

    await search(req, res, next);

    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith(mockProperties);
  });
```

```
  it('should handle empty searchQuery gracefully', async () => {
    req.query = { searchQuery: '' };

    await search(req, res, next);

    expect(res.json).toHaveBeenCalledTimes(1);
    expect(res.json).toHaveBeenCalledWith({ message: 'Could not find' });
  });
});
```

Result:

```
property search()
  √ should return properties if searchQuery is provided (2 ms)
  √ should return empty array if no properties match searchQuery (2 ms)
  √ should return a message if no searchQuery is provided (1 ms)
  √ should return an error if database query fails (48 ms)
  √ should handle invalid searchQuery gracefully (3 ms)
  √ should pass error to next middleware in case of failure (11 ms)
  √ should use regex in query for searchQuery (1 ms)
  √ should return properties with correct structure when searchQuery matches
  √ should return properties case-insensitively (1 ms)
  √ should handle empty searchQuery gracefully
```

b) BookVisitSlot():

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

  beforeEach(() => {
    req = {
      body: {
        buyerId: 'buyer123',
        sellerId: 'seller123',
        date: '2023-10-10',
        visitSlot: '10:00 AM',
        listingId: 'listing123'
      },
      params: {
```

```
        id: 'buyer123'
      },
      user: {
        id: 'buyer123'
      }
    };

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

    next = jest.fn();
  });

  // Normal Tests
  it('should book a visit slot successfully when all conditions are met',
async () => {
    User.findOne.mockResolvedValueOnce({ _id: 'buyer123' });
    User.findOne.mockResolvedValueOnce({ _id: 'seller123' });
    VisitSlot.findOne.mockResolvedValueOnce(null);
    VisitSlot.create.mockResolvedValueOnce(req.body);

    await bookVisitSlot(req, res, next);

    expect(res.status).toHaveBeenCalledWith(200);
    expect(res.json).toHaveBeenCalledWith(req.body);
  });

  // Edge Case Tests
  it('should return 404 if buyer is not found', async () => {
    User.findOne.mockResolvedValueOnce(null);

    await bookVisitSlot(req, res, next);

    expect(next).toHaveBeenCalledWith(errorHandler(404, 'User Not
Found!'));
  });

  it('should return 404 if seller is not found', async () => {
```

```
    User.findOne.mockResolvedValueOnce({ _id: 'buyer123' });
    User.findOne.mockResolvedValueOnce(null);

    await bookVisitSlot(req, res, next);

    expect(next).toHaveBeenCalledWith(errorHandler(404, 'User Not
Found!'));
  });

  it('should return 400 if date is empty', async () => {
    User.findOne.mockResolvedValueOnce({ _id: 'buyer123' });
    User.findOne.mockResolvedValueOnce({ _id: 'seller123' });
    VisitSlot.findOne.mockResolvedValueOnce(null);
    req.body.date = '';

    await bookVisitSlot(req, res, next);

    expect(next).toHaveBeenCalledWith(errorHandler(400, 'Date cannot be
empty!'));
  });

  it('should return 400 if visitSlot is empty', async () => {
    User.findOne.mockResolvedValueOnce({ _id: 'buyer123' });
    User.findOne.mockResolvedValueOnce({ _id: 'seller123' });
    VisitSlot.findOne.mockResolvedValueOnce(null);
    req.body.visitSlot = '';

    await bookVisitSlot(req, res, next);

    expect(next).toHaveBeenCalledWith(errorHandler(400, 'Visit Slot cannot
be empty!'));
  });

  it('should return 401 if user tries to book a slot for another user',
async () => {
    User.findOne.mockResolvedValueOnce({ _id: 'buyer123' });
    User.findOne.mockResolvedValueOnce({ _id: 'seller123' });
    VisitSlot.findOne.mockResolvedValueOnce(null);
    req.params.id = 'anotherUser';
```

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

    expect(next).toHaveBeenCalledWith(errorHandler(401, 'You can only book
you own Visit Slots!'));
  });


  it('should handle errors during slot creation', async () => {
    const error = new Error('Database error');
    User.findOne.mockResolvedValueOnce({ _id: 'buyer123' });
    User.findOne.mockResolvedValueOnce({ _id: 'seller123' });
    VisitSlot.findOne.mockResolvedValueOnce(null);
    VisitSlot.create.mockRejectedValueOnce(error);

    await bookVisitSlot(req, res, next);

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


});
```

Result:

```
bookVisitSlot() bookVisitSlot method
  √ should book a visit slot successfully when all conditions are met (3 ms)
  √ should return 404 if buyer is not found
  √ should return 404 if seller is not found
  √ should return 400 if date is empty (1 ms)
  √ should return 400 if visitSlot is empty (1 ms)
  √ should return 401 if user tries to book a slot for another user (1 ms)
  √ should handle errors during slot creation (1 ms)
```

Code Coverage:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
controllers	100	100	100	100	
property.controller.js	100	100	100	100	
models	100	100	100	100	
listing.model.js	100	100	100	100	
user.model.js	100	100	100	100	
visitSlot.model.js	100	100	100	100	
utils	100	100	100	100	
error.js	100	100	100	100	