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"`

Favourite Controller:

a) deleteFav();

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

        beforeEach(() => {
            req = {
                params: {
                    id: '12345'
                }
            };
            res = {
```

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

    it('should delete a favourite and return a success message',
async () => {
        // Arrange
        Favourite.findByIdAndDelete.mockResolvedValueOnce(true);

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

        // Assert

expect(Favourite.findByIdAndDelete).toHaveBeenCalledWith('12345');
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.json).toHaveBeenCalledWith('Favourite has
been deleted!');
    });

    it('should handle the case where the favourite does not
exist', async () => {
        // Arrange
        Favourite.findByIdAndDelete.mockResolvedValueOnce(null);

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

        // Assert

expect(Favourite.findByIdAndDelete).toHaveBeenCalledWith('12345');
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.json).toHaveBeenCalledWith('Favourite has
been deleted!');
    });

    it('should handle errors thrown by the database operation',
async () => {
```

```
            // Arrange
            const error = new Error('Database error');

Favourite.findByIdAndDelete.mockRejectedValueOnce(error);

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

            // Assert

expect(Favourite.findByIdAndDelete).toHaveBeenCalledWith('12345');
            expect(next).toHaveBeenCalledWith(error);
        });
    });
```

Result:

```
PASS api/controllers/test/favourite.test.js
  Favourite Controller
    deleteFav()
      √ should delete a favourite and return a success message (6 ms)
      √ should handle the case where the favourite does not exist (1 ms)
      √ should handle errors thrown by the database operation (2 ms)
```

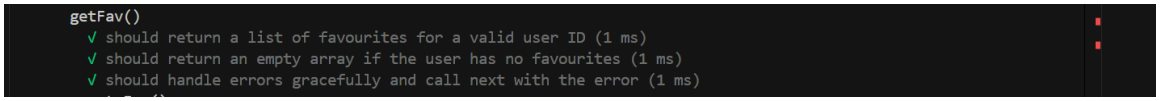
(b) getFav();

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

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

```
        it('should return a list of favourites for a valid user ID',
async () => {
            // Arrange
            const mockFavourites = [{ id: 'fav1' }, { id: 'fav2' }];
            Favourite.find.mockResolvedValue(mockFavourites);

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

            // Assert
            expect(Favourite.find).toHaveBeenCalledWith({ userId:
'user123' });
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith(mockFavourites);
        });

        it('should return an empty array if the user has no
favourites', async () => {
            // Arrange
            Favourite.find.mockResolvedValue([]);

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

            // Assert
            expect(Favourite.find).toHaveBeenCalledWith({ userId:
'user123' });
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith([]);
        });

        it('should handle errors gracefully and call next with the
error', async () => {
            // Arrange
            const error = new Error('Database error');
            Favourite.find.mockRejectedValue(error);

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

```
            // Assert
            expect(Favourite.find).toHaveBeenCalledWith({ userId:
'user123' });
            expect(next).toHaveBeenCalledWith(error);
        });
    });
```

Result:



(c) createFav();

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

        beforeEach(() => {
            req = {
                body: {
                    form1: {
                        userId: 'user123',
                        listingId: 'listing123',
                        // other fields as required by the Favourite
model
                    }
                }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should create a favourite and return it with status
201', async () => {
            // Arrange
            const favData = { ...req.body.form1, _id: 'fav123' };
            Favourite.create.mockResolvedValue(favData);
```

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

            // Assert

expect(Favourite.create).toHaveBeenCalledWith(req.body.form1);
            expect(res.status).toHaveBeenCalledWith(201);
            expect(res.json).toHaveBeenCalledWith(favData);
        });

        it('should handle validation errors gracefully', async () =>
{
            // Arrange
            const error = new Error('Validation Error');
            Favourite.create.mockRejectedValue(error);

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

            // Assert

expect(Favourite.create).toHaveBeenCalledWith(req.body.form1);
            expect(next).toHaveBeenCalledWith(error);
        });

        it('should handle unexpected errors gracefully', async () =>
{
            // Arrange
            const error = new Error('Unexpected Error');
            Favourite.create.mockRejectedValue(error);

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

            // Assert

expect(Favourite.create).toHaveBeenCalledWith(req.body.form1);
            expect(next).toHaveBeenCalledWith(error);
        });
    });
```

Result:

```
createFav()
  √ should create a favourite and return it with status 201 (1 ms)
  √ should handle validation errors gracefully
  √ should handle unexpected errors gracefully
```

(d) isFav();

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

        beforeEach(() => {
            req = {
                params: { id: 'listing123' },
                user: { id: 'user123' }
            };
            res = {
                status: jest.fn().mockReturnThis(),
                json: jest.fn()
            };
            next = jest.fn();
        });

        it('should return the favourite when it exists', async () =>
{
            // Arrange
            const fav = { userId: 'user123', listingId: 'listing123'
};
            Favourite.findOne.mockResolvedValue(fav);

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

            // Assert
            expect(Favourite.findOne).toHaveBeenCalledWith({ userId:
'user123', listingId: 'listing123' });
            expect(res.status).toHaveBeenCalledWith(200);
            expect(res.json).toHaveBeenCalledWith(fav);
        });
```

```
        it('should return 204 if user is not logged in', async () =>
{
            // Arrange
            req.user.id = null;

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

            // Assert
            expect(res.status).toHaveBeenCalledWith(204);
            expect(res.json).toHaveBeenCalledWith('User not Logged
In!');
        });

        it('should return 204 if no favourite is found', async () =>
{
            // Arrange
            Favourite.findOne.mockResolvedValue(null);

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

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

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

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

            // Assert
```

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

Result:

```
    isFav()
      ✓ should return the favourite when it exists (1 ms)
      ✓ should return 204 if user is not logged in (1 ms)
      ✓ should return 204 if no favourite is found (1 ms)
      ✓ should call next with an error if an exception is thrown (1 ms)
```

Code Coverage:

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