

**Project X – Unit Testing using Jest**

**Prepared by: Team BAMP**

**Sophie A. Bohol**

**Cloe Den Mabanding**

**Maureen Portillo**

**Jhomarie Avenido**

Unit Testing using Jest

Overview

This documentation covers the unit testing implementation for the QR-based Attendance System. The tests are implemented using Jest testing framework and focus on the core models: User, Course, and Attendance.

**Running Tests**

```bash

# Run all tests

npm test

Run tests with coverage

npm run test:coverage

Run specific test file

npm test -- tests/models/user.test.js

```

**Test Coverage**

User Model Tests (`user.test.js`)

- User Creation

- Creates and saves user successfully

- Validates required fields

- Prevents duplicate usernames

- Handles password validation

Course Model Tests (`course.test.js`)

-\*Course Management

- ✓ Creates and saves course successfully

- ✓ Validates required fields

- ✓ Prevents duplicate course codes

- ✓ Validates schedule format

Attendance Model Tests (`attendance.test.js`)

- Attendance Tracking

- ✓ Creates attendance record with QR data

- ✓ Records student attendance scan

- ✓ Prevents duplicate student scans

- ✓ Validates attendance record expiration

Test Implementation Details

Database Setup

- Uses MongoDB test database

- Cleans up test data after each test

- Handles database connections properly

Validation Testing

```javascript

// Example: Required Fields Validation

it('should fail to save user without required fields', async () => {

const userWithoutRequired = new User({

firstName: 'John'

});

let err;

try {

await userWithoutRequired.save();

} catch (error) {

err = error;

}

expect(err).toBeInstanceOf(mongoose.Error.ValidationError);

});

```

Duplicate Prevention

```javascript

// Example: Duplicate Username Check

it('should not save duplicate username', async () => {

// First user

await new User({

idNumber: '2024-0001',

username: 'johndoe',

// ... other fields

}).save();

// Duplicate username

const duplicateUser = new User({

idNumber: '2024-0002',

username: 'johndoe',

// ... other fields

});

await expect(duplicateUser.save()).rejects.toThrow();

});

```

QR Code Validation

```javascript

// Example: QR Code Attendance

it('should create attendance record with QR data', async () => {

const qrData = JSON.stringify({

courseId: courseId,

courseCode: 'ITP133',

// ... other QR data

});

const attendance = await new Attendance({

qrCodeData: qrData,

// ... other fields

}).save();

expect(attendance.\_id).toBeDefined();

expect(attendance.qrCodeData).toBe(qrData);

});

```

Best Practices Implemented

1. Test Isolation

- Each test runs independently

- Database cleanup between tests

- No shared state between tests

2. Error Handling

- Validation errors

- Duplicate entry errors

- Database operation errors

3. Data Validation

- Required fields

- Data types

- Format validation

- Business rules

4. Test Organization

- Clear test descriptions

- Logical grouping

- Consistent naming

Model Validations

User Model

- Required Fields: idNumber, firstName, lastName, email, username, password, role

- Unique Fields: username, email

- Password Hashing: Automatically hashed before save

Course Model

- Required Fields: courseCode, courseName, description, lecturerId

- Schedule Validation: Valid days (Mon-Sun), required start/end times

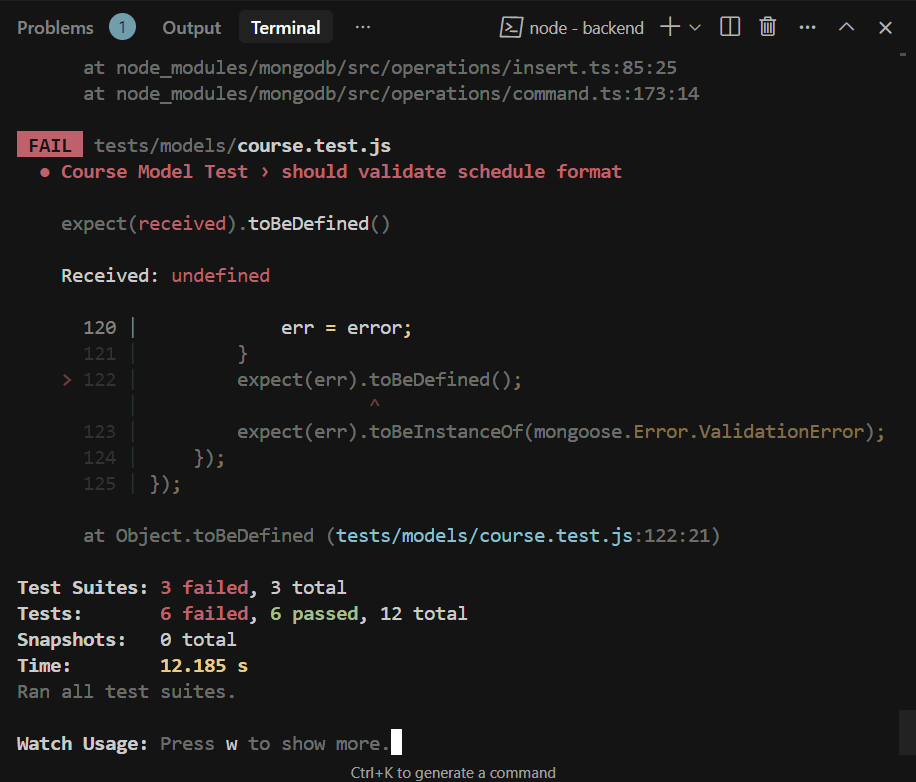
- Unique Fields: courseCode

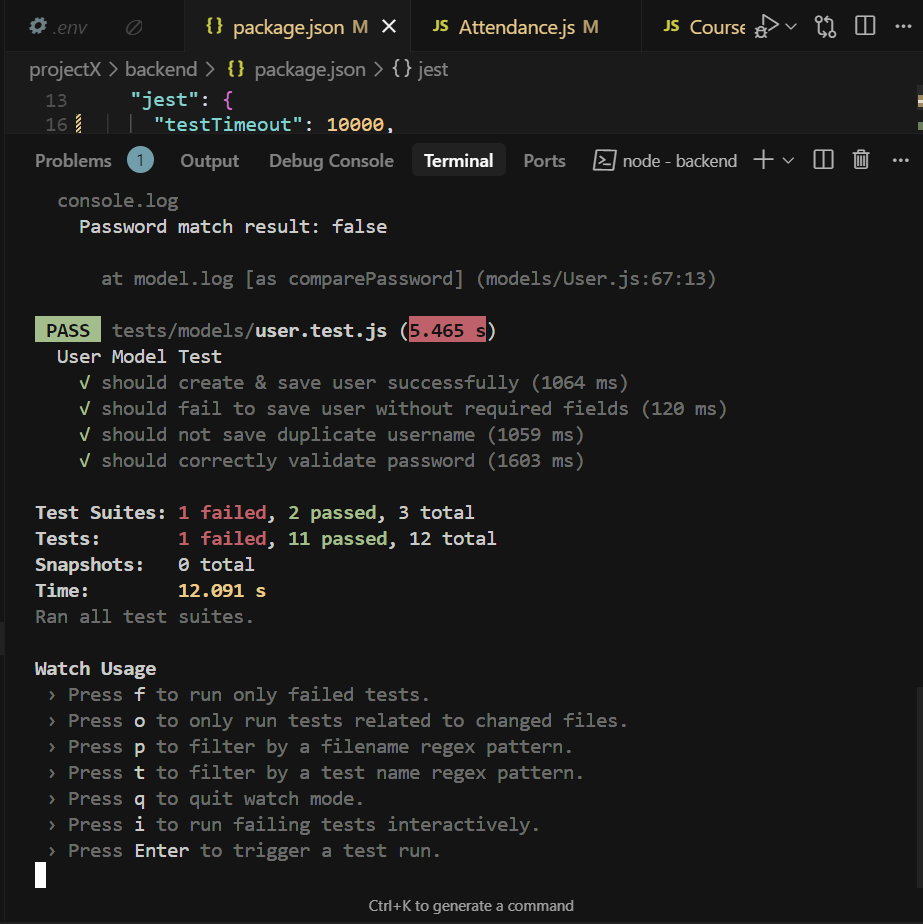
Attendance Model

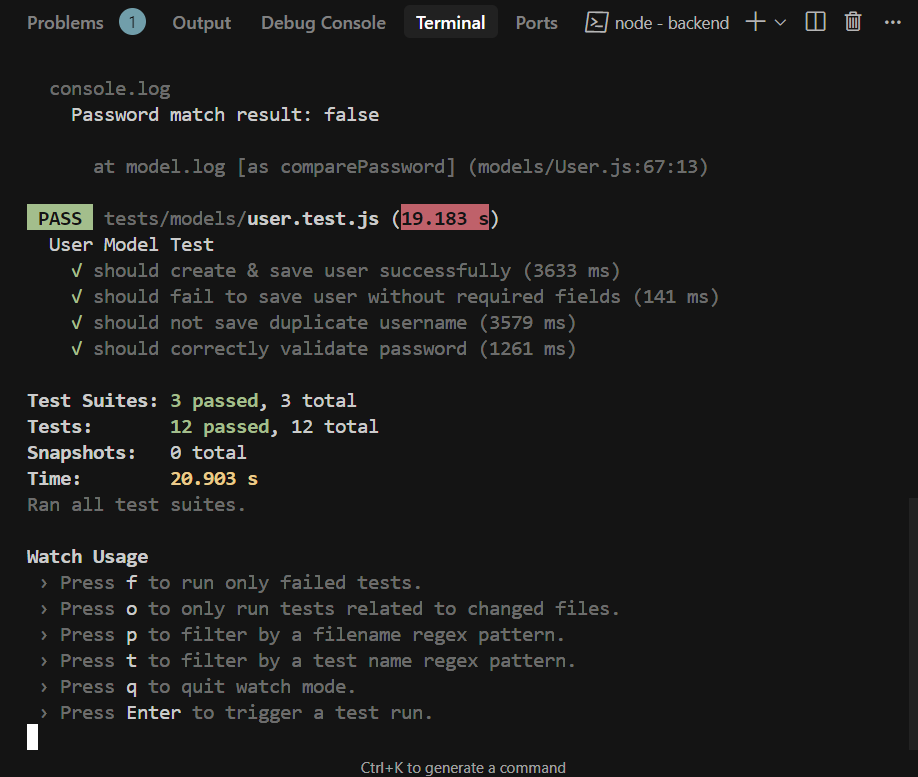
- Required Fields: courseId, lecturerId, qrCodeData, generatedAt, expiresAt

- Validation: No duplicate student scans

- Time Validation: Expiration time checks





**TEST PASSED**