TESTING POLICY

BMW IT HUB

COS 301 | University of Pretoria

FIRE-FIGHTER ACCESS MANAGEMENT PLATFORM

Table of Contents

1. Introduction

1.1 Purpose

This document outlines the comprehensive testing policy for the FireFighter Access Management Platform. It defines the procedures, tools, and standards for ensuring software quality through automated and systematic testing approaches.

1.2 Scope

This policy covers all testing activities for:

- **Backend API (FF-API)**: Spring Boot application with Java 17
- **Frontend Application (FF-Angular)**: Angular 19 with Ionic framework
- **Integration Testing**: End-to-end system testing
- **Performance Testing**: Load and stress testing with JMeter

1.3 Objectives

- Ensure high-quality software delivery through comprehensive testing
- Maintain automated testing pipelines for continuous integration
- Provide clear procedures for test execution and reporting
- Establish quality gates for deployment readiness

2. Testing Strategy

2.1 Testing Pyramid

Our testing strategy follows the testing pyramid approach:

- **Unit Tests (70%)**: Fast, isolated tests for individual components
- **Integration Tests (20%)**: Component interaction and API testing
- **End-to-End Tests (10%)**: Complete user journey testing

2.2 Testing Types

2.2.1 Backend Testing

- **Unit Tests**: Service layer, repository layer, and utility classes
- **Integration Tests**: API endpoints, database interactions, external services
- **Security Tests**: Authentication, authorization, and input validation
- **Performance Tests**: Load testing with JMeter

2.2.2 Frontend Testing

- **Unit Tests**: Components, services, and utilities
- **Component Tests**: Angular component behaviour and rendering
- **Integration Tests**: Service interactions and HTTP calls
- **E2E Tests**: User workflows and critical paths

3. Automated Testing Tools

3.1 Tool Selection and Justification

3.1.1 Jenkins CI/CD

Selected Tool: Jenkins with GitHub integration
Justification:

- **Mature Ecosystem**: Extensive plugin support and community
- **GitHub Integration**: Native support for pull request testing
- **Pipeline as Code**: Jenkinsfile for version-controlled CI/CD
- **Scalability**: Supports parallel execution and distributed builds
- **Cost-Effective**: Open-source solution suitable for academic projects

3.1.2 Backend Testing Tools

Primary Tools:

- **JUnit 5**: Modern testing framework with improved assertions
- **Mockito**: Mocking framework for isolated unit testing
- **Spring Boot Test**: Integration testing with application context
- **H2 Database**: In-memory database for fast test execution
- **JaCoCo**: Code coverage analysis and reporting

Justification:

- **Industry Standard**: Widely adopted in Spring Boot ecosystem
- **Comprehensive**: Covers unit, integration, and coverage testing
- **Fast Execution**: H2 enables rapid test cycles
- **Detailed Reporting**: JaCoCo provides actionable coverage metrics

3.1.3 Frontend Testing Tools

Primary Tools:

- **Jasmine**: Behavior-driven testing framework
- **Karma**: Test runner with browser automation
- **ChromeHeadless**: Headless browser for CI/CD environments
- **Angular Testing Utilities**: TestBed and component testing tools

Justification:

- **Angular Native**: Integrated with Angular CLI and ecosystem
- **Cross-Browser**: Karma supports multiple browser environments
- **CI/CD Friendly**: Headless execution for automated pipelines
- **Rich Assertions**: Jasmine provides expressive test syntax

3.1.4 Performance Testing Tools

Primary Tool: Apache JMeter 5.6.3 **Justification**:

- **Comprehensive**: Supports various protocols and load patterns
- **Reporting**: Detailed performance metrics and visualizations
- **Scriptable**: Command-line execution for CI/CD integration
- **Open Source**: No licensing costs for academic use

4. Testing Procedures

4.1 Pre-Development Testing

1. **Test-Driven Development (TDD)**: Write tests before implementation
2. **Test Planning**: Define test scenarios during feature planning
3. **Test Data Preparation**: Create realistic test datasets

4.2 Development Testing

1. **Unit Test Execution**: Run tests during development
2. **Local Integration Testing**: Verify component interactions
3. **Code Coverage Monitoring**: Maintain minimum coverage thresholds

4.3 Pre-Commit Testing

1. **Automated Test Execution**: All tests must pass before commit
2. **Code Quality Checks**: Linting and static analysis
3. **Coverage Validation**: Ensure coverage requirements are met

4.4 CI/CD Pipeline Testing

1. **Automated Trigger**: Tests run on every push and pull request
2. **Parallel Execution**: Unit and integration tests run concurrently
3. **Quality Gates**: Deployment blocked if tests fail
4. **Notification**: Team notified of test results via GitHub status

4.5 Release Testing

1. **Full Test Suite**: Complete test execution before release
2. **Performance Testing**: Load testing with JMeter

5. Test Repository Structure

5.1 Backend Test Organization

FF-API/src/test/

```
├── java/com/apex/firefighter/
│   ├── unit/                    # Unit tests (isolated)
│   │   ├── controllers/         # REST controller tests
│   │   ├── services/           # Business logic tests
│   │   ├── repositories/       # Data access tests
│   │   └── models/             # Entity tests
│   ├── integration/            # Integration tests
│   │   ├── api/                # End-to-end API tests
│   │   ├── database/           # Database integration
│   │   └── external-services/  # External service tests
│   └── utils/                  # Test utilities
├── resources/
│   ├── application-test.properties
│   ├── sql/                    # Test database scripts
│   └── postman/               # API test collections
└── FF-API-Test-Suite-Overview.md
```

5.2 Frontend Test Organization

FF-Angular/src/

```
├── app/
│   ├── components/
│   │   └── *.spec.ts          # Component tests
│   ├── services/
│   │   └── *.spec.ts          # Service tests
│   ├── pages/
│   │   └── *.spec.ts          # Page component tests
│   └── guards/
│       └── *.spec.ts          # Route guard tests
├── karma.conf.js              # Test configuration
└── test.ts                    # Test setup
```

5.3 Performance Test Organization

jmeter-testing/

```
├── ff-api-tests/
│   ├── test-plans/            # JMeter test plans (.jmx)
│   ├── test-data/             # Test data files
│   ├── scripts/               # Automation scripts
│   ├── reports/               # Generated reports
│   └── config.properties      # Test configuration
└── apache-jmeter-5.6.3/       # JMeter installation
```

6. CI/CD Integration

6.1 Jenkins Pipeline Configuration

Our Jenkins pipeline (Jenkinsfile) implements automated testing with the following stages:

1. **Checkout**: Source code retrieval from GitHub
2. **Install Dependencies**: Backend (Maven) and Frontend (npm) dependencies
3. **Unit Tests**: Isolated component testing
4. **Integration Tests**: Component interaction testing
5. **Frontend Tests**: Angular application testing
6. **Build**: Application compilation and packaging
7. **Quality Gates**: Coverage and quality validation

6.2 GitHub Integration

- **Status Checks**: Real-time build and test status on pull requests
- **Branch Protection**: Requires passing tests before merge
- **Notifications**: Automatic team notifications on test failures

6.3 Test Execution Commands

Backend Testing

```
# Unit Tests
mvn -Dtest=com.apex.firefighter.unit.**.*Test test -
Dspring.profiles.active=test

# Integration Tests
mvn -Dtest=com.apex.firefighter.integration.**.*Test test -
Dspring.profiles.active=test

# All Tests with Coverage
mvn clean test jacoco:report
```

Frontend Testing

```
# Unit Tests (CI Mode)
npm test --watch=false --browsers=ChromeHeadless

# Unit Tests (Development Mode)
npm test

# Linting
npm run lint
```

Performance Testing

```
# Load Testing
jmeter -n -t test-plans/load-test.jmx -l reports/results.jtl -e -o
reports/html/
```

7. Testing Coverage Requirements

7.1 Coverage Targets

- **Backend Services**: Minimum 80% line coverage
- **Backend Controllers**: Minimum 70% line coverage
- **Frontend Services**: Minimum 70% line coverage
- **Frontend Components**: Minimum 60% line coverage

7.2 Coverage Tools

- **Backend**: JaCoCo Maven plugin with XML and HTML reports
- **Frontend**: Karma coverage with Istanbul reporter

7.3 Coverage Enforcement

- **Quality Gates**: Build fails if coverage drops below thresholds
- **Trend Monitoring**: Coverage trends tracked over time
- **Exclusions**: Configuration files and generated code excluded

8. Test Reports and Documentation

8.1 Automated Reports

- **JUnit Reports**: XML format for Jenkins integration
- **Coverage Reports**: HTML reports for detailed analysis
- **Performance Reports**: JMeter HTML dashboard reports

8.2 Report Locations

- **Backend Test Reports**: FF-API/target/surefire-reports/
- **Backend Coverage**: FF-API/target/site/jacoco/
- **Frontend Test Reports**: FF-Angular/coverage/
- **Performance Reports**: jmeter-testing/ff-api-tests/reports/

8.3 Documentation

- **Test Suite Overview**: FF-API/src/test/FF-API-Test-Suite-Overview.md
- **Performance Testing**: jmeter-testing/ff-api-tests/COMPREHENSIVE_TEST_GUIDE.md
- **Load Test Results**: jmeter-testing/ff-api-tests/LOAD_TEST_RESULTS_REPORT.md

9. Quality Gates

9.1 Deployment Criteria

All of the following must pass before deployment:

- All unit tests pass (100% success rate)
- All integration tests pass (100% success rate)
- Code coverage meets minimum thresholds
- No critical security vulnerabilities
- Performance tests within acceptable limits
- Code quality checks pass (linting, static analysis)

10. Conclusion

This testing policy ensures the FireFighter Access Management Platform maintains high quality through comprehensive automated testing. The combination of Jenkins CI/CD, comprehensive test suites, and strict quality gates provides confidence in our software delivery process.

Repository Links:

- **Main Repository**: COS301-SE-2025/Fire-Fighter: Secure temporary privilege escalation system for emergency IT access. Features time-bound roles, audit logging & real-time notifications. Built with Angular + Spring Boot.
- **Test Cases**: FF-API/src/test/ and FF-Angular/src/app/**/*.spec.ts
- **Test Reports**: Generated in respective target/ and coverage/ directories
- **Performance Tests**: jmeter-testing/ff-api-tests/