FUTURE FEED

CONTACT: SYNTEXSQUAD@GMAIL.COM

FutureFeed – Testing Policy Document

1. Introduction

The purpose of this testing policy is to ensure the quality, reliability, and maintainability of the FutureFeed social media platform. Testing forms a critical component of our development lifecycle, ensuring that new features meet requirements, existing functionality remains stable, and deployments can occur with confidence.

This policy describes:

- The testing tools and framework chosen.
- The testing methodology followed
- The automation setup(CI/CD pipeline)
- References to our Github repository containing test cases and test reports.

2. Choice of Testing Tools and Justification

JUnit 5 (Jupiter):

Provides a modern and flexible framework for writing unit and integration tests in Java. Its annotations and assertions are well-suited for Spring Boot projects.

Spring Boot Test + MockMvc:

Allows for integration testing of controllers and services in isolation, simulating HTTP requests without deploying the application on an actual server.

Mockito:

Enables mocking of dependencies (e.g., repositories, external services), ensuring unit tests remain isolated.

H2 In-Memory Database:

Used during integration tests to simulate the PostgreSQL database while allowing rapid test execution and automatic cleanup.

Maven Surefire and Failsafe plugins:

Surefire runs unit tests (e.g., *Test.java).

Failsafe runs integration tests (e.g., *IT.java).

This separation ensures fast developer feedback (unit tests) and more comprehensive system checks (integration tests).

GitHub Actions CI (with option for Travis CI):

GitHub Actions was selected as the primary CI/CD tool because it integrates directly with our GitHub repository, provides free build minutes, and offers flexible workflows.

Travis CI was considered for legacy compatibility, but GitHub Actions is better aligned with our workflow.

3. Testing Procedures

3.1 Unit Testing

Each service method is tested in isolation using Mockito.

Focus is on verifying:

- Business logic correctiness
- Repository interactions(save, delete, search).
- Example: PostServicePost validates post creation, deletion, search, and tagging topic

3.2 Integration Testing

Conducted using Spring Boot Test and MockMvc. Uses H2 in-memory DB and OAuth2 login simulation for realistic flows.

Covers:

- REST endpoints (/api/posts, /api/comments, etc.).
- Database persistence with JPA repositories
- End-to-end behaviour (request → service → database → response).
- Example: PostIT validates creating, retrieving, deleting, and searching posts via actual HTTP requests.

3.3 Automated Pipeline

Trigger: Tests run automatically on every push and pull request.

Steps:

- Checkout code.
- Set up JDK 21.
- Cache Maven dependencies.
- Run Unit Tests (Surefire).
- Run Integration Tests (Failsafe).
- Failing tests will prevent merging/deployment, ensuring only stable builds are promoted.

3.4 Deployment Testing

Once all tests pass, the backend can be packaged into a JAR and deployed to AWS EC2.

Future integration: add automated smoke tests post-deployment.

4. Repository of Tests

GitHub Repository: FutureFeed GitHub Repository

Test Files:

- Unit Tests: src/test/java/com/syntexsquad/futurefeed/PostServiceTest.java, etc.
- Integration Tests: src/test/java/com/syntexsquad/futurefeed/PostIT.java, etc.

Test Reports:

Generated automatically by Maven (under target/surefire-reports and target/failsafe-reports) and viewable in GitHub Actions logs.

5. Reporting and Coverage

Developers review test results via GitHub Actions after each commit.

Maven reports include:

- JUnit XML Reports: Machine-readable for CI.
- Console Logs: Human-readable for debugging failures.
- Coverage tooling (JaCoCo) will be integrated in future iterations to monitor test coverage across modules.

6. Conclusion

This testing policy ensures:

- High-quality code through systematic unit and integration testing.
- Automation of testing and deployment via GitHub Actions.
- Transparency of results via linked test cases and reports.

By adhering to this policy, FutureFeed maintains a stable, secure, and scalable backend that supports ongoing feature development and reliable deployments.