Test Strategy Template

# 1. Scope

# This document defines the testing strategy for the Inventory Management System developed in Java using IntelliJ IDEA. The scope of testing includes functionalities like product management (add, update, delete), stock viewing, price-based filtering, inventory value calculation, and exporting data. Emphasis is placed on both unit testing and mutation testing to ensure correctness and robustness of business logic. 2. Testing Types

- **Unit Testing** using JUnit 5.

- **Mutation Testing** using custom mutant classes to validate test coverage and effectiveness.

- **Integration Testing** for database interactions via JDBC (SQLite in-memory DB).

- **System Testing** to validate end-to-end workflow logic.

# 3. Testing Approach

- **Manual Testing** for CLI-based user interactions and output verification.

- **Automated Testing**:

* JUnit 5 framework for test execution.
* Mutation testing through mutant classes (mutant1, mutant2, etc.) to detect untested or poorly-tested code paths.

- **Mocking/Stubbing** used where necessary (e.g., using Mockito for simulating behaviors).

- **Output validation** is done via redirection and comparison of System.out.

# 4. Test Environment

- **Development IDE:** IntelliJ IDEA

- **OS:** Windows 10

- **Java Version:** Java 17 (or applicable)

- **Database:** SQLite (in-memory)

- **Libraries Used:** JUnit 5, Mockito, JDBC

# 5. Test Deliverables

-Test Plan Document

- Unit Test Classes (inventorymanagementsystemTest.java)

- Mutation Test Reports (for each mutant class)

- Test Execution Logs and Console Output Captures

- Final Test Summary Report with pass/fail details and mutant survival analysis

# 6. Roles and Responsibilities

- **Test Manager:** Define test scope, select tools (JUnit, mutation testing), and manage overall testing effort.

- **Test Engineers:**

* Write unit tests for all public methods.
* Create and run mutation tests using variants like mutant1, mutant2, etc.
* Capture coverage and survival statistics.

- **Developers:**

* Fix failing test cases or refine logic where mutants survive.
* Provide clarification on logic if test engineers report ambiguous behavior.

# 7. Risks and Mitigation

**Risk**: Mutation tests survive due to insufficient assertions  
**Mitigation**: Improve assertion strength and test edge cases

# 8. Schedule

- Test Planning: 1/5 – 3/5  
- Unit Test Development: 4/5 – 7/5  
- Mutation Test Creation and Execution: 8/5  
- Final Report: 9/5