COMPREHENSIVE TEST PLAN

Fitness Reservation System

---

1. Executive Summary

This document describes the comprehensive test strategy applied to the Fitness Reservation System. The testing approach follows a **shift-left testing philosophy**, combining Test-Driven Development (TDD), automated regression testing, and Continuous Integration (CI). The system has been validated across multiple quality dimensions, including functional correctness, combinatorial input coverage, property-based validation, performance under load, security posture, and basic resilience through chaos engineering experiments in a Dockerized environment.

---

2. Test Objectives and Strategy

The primary objectives of testing are:

- To ensure **correct and deterministic pricing behavior**
- To guarantee **reservation consistency under capacity and duplication constraints**
- To validate **API correctness and robustness**
- To assess **performance, security, and resilience under adverse conditions**

Testing activities emphasize automation, reproducibility, and alignment with test engineering techniques discussed in the course.

2.1 Testing Pyramid Implementation

A layered testing pyramid was implemented:

- **Unit Tests (Base Layer)**
  Core business logic for pricing and reservations validated using TDD.

- **Integration Tests**
  API endpoints tested with in-memory and persistence-backed repositories.

- **System / End-to-End Tests**
  Realistic user journeys executed via Postman collections and Newman.

- **Cross-Cutting Tests**
  Performance testing using k6, security testing using OWASP ZAP, and resilience testing using chaos engineering tools.

This structure balances fast feedback with end-to-end confidence.

---

3. Combinatorial Testing (Pairwise Methodology)

To maximize input coverage while minimizing test count, **pairwise combinatorial testing** was applied to the pricing engine.

3.1 PICT-Based Test Generation

The following input parameters were identified:

- **Sport Type**: Yoga, Pilates, Spinning, HIIT, Zumba

- **Membership Type**: Standard, Student, Premium

- **Peak Flag**: true, false

- **Occupancy Level**: Low, Mid, High

The PICT tool was used to generate a pairwise test matrix covering all two-way interactions between parameters.

3.2 Implementation

The generated combinations were exported as CSV files and executed as **data-driven parameterized unit tests**. Expected results were computed using a golden-reference pricing tool to ensure deterministic verification.

---

4. Functional Testing

4.1 Pricing Engine (TDD)

Methodology: Red–Green–Refactor

Validated Scenarios:

- Correct application of membership multipliers
- Monotonic pricing behavior with increasing occupancy
- Peak vs. off-peak pricing behavior
- Rejection of malformed or invalid inputs

4.2 Reservation Management

Key Rules Validated:

- Session capacity cannot be exceeded
- Duplicate reservations by the same member are prevented
- Reservations for past sessions are rejected
- Unauthorized access to reservation endpoints is blocked

These rules were validated using example-based, decision-table-driven, and property-based tests.

---

5. Automated API and Integration Testing

5.1 Postman and Newman

A Postman collection was created covering authentication, session listing, and reservation workflows.

- **Automation:** Executed using Newman within the CI pipeline
- **Assertions:** HTTP status codes (200, 201, 400, 401, 409) and response structure
- **Reporting:** JUnit-style reports published as CI artifacts

This ensures API regressions are detected early and consistently.

---

6. Performance and Security Testing

6.1 Performance Testing (k6)

Performance testing was conducted using k6 with a realistic user journey scenario including login, session retrieval, and reservation attempts.

Configuration:

- Multiple concurrent virtual users
- Sustained execution suitable for CI constraints

Metrics Collected:

- Average and p95 response times
- Error rate under load

Results showed stable behavior under concurrent usage with acceptable latency and low error rates.

6.2 Security Testing (OWASP ZAP)

Security testing was performed using **OWASP ZAP baseline (passive) scan and also active scan**, integrated into the CI pipeline.

Results:

- No high or critical vulnerabilities detected
- Informational findings related to HTTP security headers and caching

Mitigations included enforcing HttpOnly and SameSite cookie policies and strict server-side validation.

---

7. Resilience and Chaos Engineering

The system was deployed in a **Dockerized environment**, enabling basic chaos engineering experiments.

7.1 Chaos Experiments (Pumba)

Chaos testing was performed using **Pumba** to inject controlled failures during runtime.

Experiments Conducted:

- Random termination of the API container during active load tests
- Observation of system behavior during temporary service unavailability

7.2 Observations

- During container termination, requests failed as expected
- Upon container restart, the service recovered automatically
- No data corruption was observed in the SQLite database
- Subsequent requests succeeded after recovery

These experiments demonstrate basic resilience and highlight areas for future improvement, such as retry mechanisms and more graceful error handling.

---

8. CI/CD Pipeline Infrastructure

All testing activities are fully automated using **GitHub Actions**:

- Build validation
- Unit and integration test execution
- Coverage report generation (HTML artifacts)
- Postman/Newman execution
- OWASP ZAP baseline scan
- k6 performance testing
- Docker-based execution and Pumba chaos experiments

This pipeline ensures consistent quality gates and reproducible results.

---

9. Conclusion

The Fitness Reservation System was validated using a comprehensive and modern testing strategy aligned with test engineering best practices. The combined use of TDD, combinatorial testing, property-based testing, automated API validation, performance testing, security scanning, and chaos engineering provides strong confidence in the system's correctness, robustness, and operational resilience.