

Test Strategy Plan

§ End-to-End Test Engineering for an Appointment and Dynamic Pricing Service

Project: Fitness Centre REST API

Author: Mehmet Kaplan

Date: 26 November 2025

Version: 1.0

§ 1. Introduction

The objective of this document is to define the test strategy for the "**Fitness Centre Appointment & Dynamic Pricing Service**". This REST-based application allows members to book group classes (yoga, spinning, pilates, etc.) and calculates prices dynamically based on membership types, occupancy rates, and peak hours.

The testing approach aims to verify functional correctness, ensure code quality through rigorous coverage metrics, and validate non-functional requirements such as performance and security. The system is built using **Java 17** and **Spring Boot 3**.

§ 2. Test Levels

The following test levels will be applied to ensure end-to-end quality assurance:

- **Unit Testing:** Focuses on testing individual components (Service layer business logic, Pricing Rules) in isolation using mocking for external dependencies.
- **Integration Testing:** Verifies the interaction between modules, specifically the Controller-Service-Repository flow and database constraints using an in-memory database (H2).
- **System / API Testing:** Black-box testing of the REST API endpoints using Postman to ensure correct HTTP responses and payload handling.
- **Performance Testing:** Load testing to evaluate system behavior under concurrent user requests (e.g., 50+ users booking simultaneously).

- **Security Testing:** Dynamic Application Security Testing (DAST) to identify vulnerabilities like injection flaws or missing security headers.

§ 3. Tools & Frameworks

The project will utilize the following tools and libraries:

Category	Tool/Framework	Purpose
Development	Java 17, Spring Boot 3, Maven	Core application framework
Unit Testing	JUnit 5, Mockito	Isolated component testing with mocking
Code Coverage	JaCoCo	Line and branch coverage measurement
Mutation Testing	PITest	Test effectiveness analysis
Property-Based Testing	jqwik	Automated input generation for invariant testing
Combinatorial Testing	ACTS (NIST)	Pairwise test case generation
API Automation	Postman & Newman	REST API functional testing
Performance Testing	k6 (or JMeter)	Load and stress testing
Security	OWASP ZAP	Vulnerability scanning (DAST)
CI/CD & Containerization	GitHub Actions, Docker	Automated testing pipeline and deployment

§ 4. Coverage Goals

To ensure high maintainability and reliability, the project aims for the following strict coverage metrics:

- **Line Coverage:** > 80%
- **Branch Coverage:** > 70%
- **Mutation Score:** Analysis of survived mutants will be performed to improve test quality and eliminate weak tests.

These targets will be continuously monitored through JaCoCo reports integrated into the CI/CD pipeline. Any decrease below the threshold will trigger a pipeline failure.

§ 5. Test Environment

The testing strategy employs multiple environments to isolate concerns and ensure comprehensive validation:

- **Local Environment (Unit Tests):** Tests run on the local development machine using Mockito to bypass the database and external dependencies.
- **Local Environment (Integration Tests):** Tests run using H2 In-Memory Database to verify data persistence, entity relationships, and repository methods without requiring external database setup.
- **CI/CD & System Environment:** The application runs in a Docker Container connected to a PostgreSQL container. Performance and Security scans are executed against this containerized environment to simulate production-like conditions.

§ 6. Exit Criteria (Success Definition)

The testing phase will be considered complete and successful when the following criteria are met:

- All functional requirements (Member Management, Class Management, Reservation Management, Dynamic Pricing) are implemented and verified.
- Coverage targets (80% Line Coverage / 70% Branch Coverage) are achieved and validated by JaCoCo reports.
- All automated Postman tests pass successfully in the CI pipeline with 100% success rate.
- No critical or high-severity security vulnerabilities are detected by OWASP ZAP scanning.
- Performance testing demonstrates acceptable response times (< 500ms average) under expected load (50 concurrent users).

- All survived mutants from PITest have been analyzed and addressed with improved test cases.
-

CEN315 - Introduction to Test Engineering

Document Version 1.0 | Confidential | © 2025