***TEST PLAN***

SPRINT 1 WEEK 1

**Community Football League Manager**

**1. Introduction**

This document defines the testing strategy for the **Community Football League Manager** web application. The goal is to ensure that all features (team/player management, fixtures, standings, results, disciplinary actions, and reports) meet functional and non-functional requirements.

**2. Scope**

* **In Scope**:
  + Authentication & Role-based access (League Manager, Coach, Referee, Player, Fan).
  + CRUD operations for **Teams, Players, Fixtures, Results, Disciplinary Actions**.
  + Match scheduling & conflict detection.
  + Standings table (aggregated query).
  + Player eligibility checks.
  + Fan interactions (view standings, vote player of the week).
  + Sponsor visibility in UI.
* **Out of Scope** (for this version):
  + Mobile push notifications.
  + Advanced analytics (beyond standings + simple reports).
  + Payment processing.

**3. Test Objectives**

* Verify correctness of core workflows (team registration, match scheduling, results submission).
* Ensure data integrity (no invalid standings or duplicate fixtures).
* Validate role-based access control (e.g., only referees submit results).
* Confirm stability & reliability with automated tests in CI/CD pipeline.

**4. Test Items**

The following modules will be tested:

Auth Service – login, signup, role-based restrictions.

Team & Player Management – registration, updates, eligibility validation.

Fixture Management – scheduling, clash detection, referee assignment.

Match Results & Standings – score submission, standings table auto-update.

Disciplinary Actions – yellow/red cards, suspensions, eligibility checks.

Reporting/Analytics – standings table, end-of-season summary.

**5. Testing Approach**

**a. Unit Testing**

Focus: models, services, utility functions.

Example: Player eligibility function returns correct status (injured, suspended, cleared).

**b. Integration Testing**

Focus: API endpoints (FastAPI + SQLAlchemy).

Example: POST /fixtures/ creates fixture → GET /fixtures/ returns same fixture.

**c. End-to-End (Scenario) Testing**

Focus: User workflows.

Example: Referee submits match result → standings update → fans see new table.

**d. Non-Functional Testing**

Performance: check queries for standings scale with 50+ teams.

Security: JWT/session validity, password hashing, unauthorized access tests.

**6. Test Deliverables**

Test Plan (this document).

Test Cases (in tests/ directory).

Automated Tests (pytest + coverage).

Test Reports (pytest + CI artifacts).

Release Checklist (green CI, tagged release, docs ready).

**7. Entry & Exit Criteria**

Entry Criteria:

Features are developed & merged.

Seed data available.

Dev environment stable.

Exit Criteria:

≥90% unit test coverage.

All integration tests pass in CI.

No critical/major bugs remain.

Feature-complete for Sprint milestone.

**8. Test Environment**

Backend: FastAPI (Python 3.10+)

DB: SQLite (dev), PostgreSQL (prod-ready)

Tools: pytest, requests, httpx, SQLAlchemy

CI/CD: GitHub Actions (linting, tests, coverage)

**9. Risks & Mitigation**

Risk: Fixtures clash logic may fail with edge cases → Mitigation: add unit + integration tests for overlapping dates.

Risk: Standings aggregation could be slow → Mitigation: test with large seed dataset.

Risk: Role-based auth bypass → Mitigation: enforce + test role-permissions in every endpoint.

**10. Schedule (Aligned to Sprints)**

Sprint 1: CI setup, sample test (Teams CRUD), smoke test pipeline.

Sprint 2: Full test suite for CRUD, standings, eligibility.

Sprint 3: Error handling, validation, reporting tests, release tests, final demo.

**11. Approval**

QA/Release Engineer: Tehilla

Product Owner: Rodaine

Tech Lead: Amanda