NBA Data Analysis Project
Quality Assurance Plan Document

List of Contributors

- **Tunahan Oğuz** (Backend Developer / Tester)
- **Beyzanur Zeybek** (Requirements Analyst / Frontend Developer)
- **Alkım Doryan** (Project Manager / Tester / Scrum Master)
- **Ali Eren Kurt** (Frontend Developer / Product Owner)

Task Matrix

Task ID	Task Description	Responsible	Completion Status	Notes
Q1	Draft QA Plan structure & sections	Alkım, Beyzanur	Completed	Followed PA2 guidelines
Q2	Outline QA strategy & testing methodologies	Tunahan, Ali Eren	Completed	Unit, integration, usability, and acceptance testing
Q3	Define quality factors & metrics	All team members	Completed	Chose performance, security, usability, maintainability
Q4	Create detailed test plan & cases	Tunahan, Beyzanur	Completed	Developed 5 test cases covering major functionality
Q5	Determine bug tracking approach & tool	Alkım, Ali Eren	Completed	GitHub Issues as the bug tracker
Q6	Format final QA Plan	Beyzanur, Alkım	Completed	Ensured clear layout and references
Q7	Final review & approval	All team members	Completed	Verified alignment with project's quality requirements

Table of Contents

1. Quality Assurance Strategy

1.1 Overview

The QA strategy focuses on early detection of defects via systematic testing and reviews at each phase of development. We ensure that the platform meets functional and non-functional requirements by combining automated and manual testing.

Key objectives:

- Verify data correctness and cleanliness (especially from Kaggle sources).
- Ensure the web application is stable, secure, and performant under expected loads.
- Validate that the user interface is intuitive and meets usability standards.

1.2 Testing Methodologies

- **Unit Testing:** Each critical function or module (e.g., data cleaning, REST endpoints) will be tested in isolation with frameworks like pytest or Django's test suite.
- **Integration Testing:** Test how modules interact (e.g., ensuring the data ingestion module populates the database correctly, and the dashboard consumes the data as expected).
- **System Testing:** Validate the entire flow end-to-end (from data ingestion to final user reports).
- **Usability Testing:** Conduct user observations to gather feedback on UI clarity, navigation, and overall user experience.
- **Acceptance Testing:** Check final readiness: does the application satisfy the original Requirements Document.

1.3 Automated vs. Manual Testing

Automated

- **Unit tests** for backend services.
- **Integration tests** covering API endpoints.
- **Continuous Integration (CI)** pipeline in GitHub Actions to run tests on each commit.

Manual

- **UI/UX** testing to gather real user feedback on workflows and visual design.
- **Exploratory** testing to find edge cases not covered by automated scripts.
- **Acceptance** test sign-off by the Product Owner.

2. Quality Factors & Metrics

We define four primary quality factors, each with a measurable metric:

Quality Factor	Description	Measurement Metric
Performance	Response time for user actions and data queries	Average response time (ms) under normal load
Integration Efficiency	Effectiveness of data integration across multiple sources	Average query execution time for integrated datasets
Maintainability	Ease of modifying and extending the codebase	Cyclomatic complexity or similar maintainability index
Data Accuracy	Correctness and consistency of the processed data	% of accurately processed records after ingestion
Report Generation Speed	Quick generation and availability of requested reports	Average time (seconds) to generate reports (PDF/CSV)

Performance: Aim for less than **2 seconds** average response time.

Maintainability: Keep cyclomatic complexity below an agreed threshold (e.g., <15 for core functions).

Data Accuracy: Ensure at least **98% accuracy** in data ingestion and cleaning processes.

Integration Efficiency: Achieve an average integrated query execution time below **500ms**.

Report Generation Speed: Generate user-requested or scheduled reports within **40 seconds**.

---

3. Test Plan

3.1 Test Cases

Below are **5 representative test cases** that illustrate the QA approach:

1. **TC-1: Data Ingestion Validation**
- **Objective**: Ensure that CSV files (e.g., common_player_info.csv) are correctly read, cleaned, and loaded into the database.
- **Preconditions**: CSV file present with valid data format.
- **Steps**:
    1. Trigger ingestion process via script or API.
    2. Verify logs for any ingestion errors.
    3. Check database for inserted records.
- **Expected Result**: No errors; correct number of records in the database

2. **TC-2: Dashboard Visualization**
- **Objective**: Validate that the main dashboard loads and interactive charts respond to filters.
- **Preconditions**: Database is populated with data.
- **Steps**:
    1. Launch the web app and navigate to the dashboard.
    2. Apply a filter (e.g., select a specific player or team).
    3. Observe chart updates.
- **Expected Result**: Dashboard loads within 2 seconds, charts update dynamically without errors.

3. **TC-3: Automated Report Generation**
- **Objective**: Confirm the system generates a PDF report of selected player stats.
- **Preconditions**: Database has valid data, user is logged in.
- **Steps**:
    1. Navigate to “Reports” section and select “Generate PDF”.
    2. Wait for back-end processing.
    3. Download the resulting PDF.
- **Expected Result**: PDF file downloads with accurate and properly formatted data.

4. **TC-4: Data Export (CSV)**

- **Objective**: Validate that a user can export filtered data (e.g., top 10 scorers) to a CSV file.
- **Preconditions**: Database is populated; user has selected a data subset.
- **Steps**:

  1. Choose “Export CSV” in the user interface.
  2. Wait for download prompt.
  3. Inspect the downloaded CSV.

- **Expected Result**: CSV matches the selected dataset, columns and rows are correct.

5. **TC-5: Security & Role Management**

- **Objective**: Test that only authorized users can access admin functionalities.
- **Preconditions**: User roles are predefined in the system (Admin vs. Regular).
- **Steps**:

  1. Login as a regular user.
  2. Attempt to access an admin-only endpoint.

- **Expected Result**: Access is denied, and an appropriate error message is shown.

3.2 Bug Tracking

We will use **GitHub Issues** for bug tracking:

- Each bug gets a unique issue with a clear title and description.
- Severity labels (Critical, Major, Minor) to prioritize resolution.
- Automatic notifications to assignees; status updated as issues move from Open → In Progress → Resolved → Closed.