Sprint 1 / MVP Requirements: C++ Match Ticketing System

Project: Sports Match Ticketing System
Sprint: 01 (MVP Launch - Walking Skeleton)
Methodology: Agile
Goal: Establish core data structure, storage, and the primary "Happy Path" for Administrators and Fans.

1. Executive Summary

The objective is to deliver a Minimum Viable Product (MVP) console application. By the end of this sprint, an administrator must be able to define match inventory, and a fan must be able to view availability and reserve a seat. All data must persist to a single CSV file (matches.csv) between application restarts.

Scope:

- **CLI:** Single-word inputs only (no spaces) to simplify input handling.
- **Roles:** Admin (Add Inventory) vs. Fan (View & Book).
- **Core Logic:** Direct inventory management (no individual Ticket generation).

2. Core User Stories (Backlog)

US-01: Admin - Initialize Inventory (FR-MATCH-01)

As an Administrator, **I want to** add a new match code and capacity, **So that** tickets are available to sell.

- **Inputs:** Match Code (e.g., "LIV-CHE"), Total Capacity (int).
- **Constraint:** Inputs must be single words (no spaces).
- **Output:** Confirmation message "Inventory Initialized".
- **Technical:** Appends new row to matches.csv.

US-02: Fan - View & Reserve (FR-BOOK-01)

As a Fan, **I want to** view match availability and reserve a seat, **So that** I can secure a spot before it sells out.

- **View Logic:** Display Match ID, Code, and Available Seats (Capacity - Booked).
- **Booking Logic:**
    1. User inputs Match ID.
    2. System checks if Booked < Capacity.
    3. **Valid:** Increments Booked counter in memory and updates matches.csv.
    4. **Invalid:** Displays "Sold Out" or "Invalid ID".

(Note: US-04 "Persistence" has been merged into the Acceptance Criteria below).

3. Technical Specifications

3.1. Data Structures (C++ Class)

One simplified class to avoid pointer/memory complexity:

- **Class Match:**
  - int id (Unique Identifier)
  - string matchCode (e.g., "Real_vs_Barca")
  - int capacity
  - int booked (Counter starts at 0, increments on reservation)

3.2. Storage Schema (matches.csv)

The file serves as the database. It is loaded on startup and updated on every modification.

```
ID,MatchCode,Capacity,Booked
1,Liverpool_vs_Chelsea,54000,10
2,Real_vs_Barca,80000,500
```

4. Acceptance Criteria (Definition of Done)

A User Story is "Done" only when:

1. **Functionality:**
   - [ ] Admin can successfully add a match (e.g., "LIV-CHE", 100).
   - [ ] Fan can view the correct "Available" count.
   - [ ] Fan cannot book if Booked == Capacity (Boundary Test).
   - [ ] Fan cannot book a non-existent ID.
2. **Persistence:**
   - [ ] Application saves to CSV immediately after Admin adds a match or Fan books a seat.
   - [ ] Data is correctly reloaded after closing and reopening the app.
3. **Stability:**
   - [ ] Code compiles with no warnings/errors.
   - [ ] No crashes on standard string inputs (no spaces).

5. Sprint Tasks

1. **Setup:** Initialize C++ Project.
2. **Backend:** Create Match class and matches.csv handler (load/save).
3. **Frontend:** Build Main Menu (looping switch: Admin, Fan, Exit).
4. **Integration:** Connect Admin input to Match creation & Save.
5. **Integration:** Connect Fan input to Booked counter increment & Save.
6. **Testing:** Manual verification of persistence and sold-out logic.