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

**Project**: Sports Match Ticketing System

**Sprint**: 01 (MVP Launch - Core Backend)

**Methodology**: Agile

**Goal**: Establish the backend object model, persistent storage, and core booking logic (Headless/No UI).

## 1. Executive Summary

The objective of this sprint is to deliver the "Headless" backend logic for the ticketing system. This foundational layer will serve as the engine for future user interfaces. By the end of this sprint, the system must support defining matches and generating individual tickets, with all data persisting to relational CSV files to ensure business continuity.

**Scope:**

* **System Boundary:** Backend Logic Only (No CLI/GUI).
* **Core Logic:** Inventory management (Match) and Ticket generation (Ticket).
* **Architecture:** Manager Pattern (MatchManager, TicketManager).
* **Data Persistence:** Relational CSV storage (matches.csv, tickets.csv).

## 2. Stakeholders

* **Administrator:** Responsible for initializing match inventory.
* **Fan:** The end-user who wishes to secure a seat.
* **System Architect:** Concerned with data integrity, persistence, and code structure.

## 3. Functional Requirements (FR)

These requirements define *what* the system must do. Each is analyzed using the **SMART** framework.

### FR-01: Inventory Initialization

* **User Story:** As an Administrator, I want to programmatically add a new match fixture so that the system has inventory available for booking.
* **Functionality:** The system shall accept a Match Code and Capacity, validate them, and store the new match.
* **SMART Analysis:**
  + **Specific:** The MatchManager::addMatch method must accept string matchCode and int capacity.
  + **Measurable:** A new row must be appended to matches.csv containing the generated ID.
  + **Achievable:** Utilizes standard C++ file streams (std::ofstream) and object instantiation.
  + **Relevant:** Without inventory, tickets cannot be sold.
  + **Time-bound:** Core feature for Sprint 1 completion.
* **Input:** Match Code (e.g., "LIV-CHE"), Capacity (e.g., 5000).
* **Output:** Unique Match ID (int).

### FR-02: Ticket Reservation

* **User Story:** As a Fan, I want to request a ticket for a specific match so that I secure a unique seat.
* **Functionality:** The system shall generate a Ticket object linked to a Match if and only if seats are available.
* **SMART Analysis:**
  + **Specific:** Input requires valid matchId and ownerName. Output is a unique ticketId.
  + **Measurable:** The bookedCount for the specific match increments by exactly 1.
  + **Achievable:** Logic is a simple conditional check (booked < capacity).
  + **Relevant:** Represents the primary revenue generation mechanism.
  + **Time-bound:** Core feature for Sprint 1 completion.
* **Logic:**
  1. Validate matchId exists.
  2. Check Match::isFull().
  3. If valid, create Ticket and save.
* **Output:** Unique Ticket ID (or error).

## 4. Non-Functional Requirements (NFR)

These requirements define *how* the system performs or behaves.

### NFR-01: Data Persistence (Reliability)

* **Requirement:** The system shall persist all state changes to secondary storage immediately to prevent data loss.
* **Constraint:** Data must be stored in Comma-Separated Values (CSV) format.
* **Trigger:** Save operations must occur synchronously after every successful addMatch or bookTicket.
* **Recovery:** On startup, the MatchManager and TicketManager must reconstruct all objects and counters from the files.

### NFR-02: Data Integrity (Safety)

* **Requirement:** The system must enforce logical constraints to prevent corruption.
* **Constraints:**
  + **Boundary:** bookedCount must never exceed capacity.
  + **Referential Integrity:** A ticket cannot be created for a non-existent matchId.
  + **Uniqueness:** IDs must be unique across the system lifecycle.

### NFR-03: Code Structure (Maintainability)

* **Requirement:** The codebase must adhere to Object-Oriented principles.
* **Constraints:**
  + **Encapsulation:** Data fields (e.g., capacity) must be private, accessible only via public methods.
  + **Separation of Concerns:** Match logic is separate from Ticket logic.
  + **Pattern:** Use "Manager" classes to handle collections and file I/O.

## 5. Technical Specifications

### 5.1. Domain Classes

**Class: Match**

* **Attributes:**
  + private int id (Unique)
  + private string matchCode
  + private int capacity
  + private int bookedCount
* **Methods:**
  + + bookSeat(): bool
  + + getMatchCode(): string
  + - isFull(): bool

**Class: Ticket**

* **Attributes:**
  + private int ticketId (Unique)
  + private int matchId (Foreign Key)
  + private string ownerName
* **Methods:**
  + + getTicketId(): int
  + + getOwnerName(): string
  + + setOwnerName(name: string): void

### 5.2. Manager Classes (Controllers)

**Class: MatchManager**

* **Responsibilities:**
  + Maintain a std::vector<Match> or std::map<int, Match>.
  + Load/Save matches.csv.
* **Key Method:** + addMatch(code: string, capacity: int): int

**Class: TicketManager**

* **Responsibilities:**
  + Maintain a std::vector<Ticket>.
  + Load/Save tickets.csv.
* **Key Method:** + bookTicket(matchId: int, owner: string): int

### 5.3. Storage Schema

**matches.csv**

ID,MatchCode,Capacity,Booked  
1,LIV-CHE,54000,1

**tickets.csv**

TicketID,MatchID,OwnerName  
1001,1,JohnDoe

## 6. Acceptance Criteria (Definition of Done)

### Functional Verification

* [ ] **Inventory:** MatchManager::addMatch("LIV-CHE", 100) successfully returns a new ID and updates the CSV.
* [ ] **Booking:** TicketManager::bookTicket(matchId, "User") returns a valid Ticket ID and increments the booked count in matches.csv.
* [ ] **Constraint:** Booking fails (returns -1 or false) if bookedCount equals capacity.
* [ ] **Constraint:** Booking fails if matchId is not found.

### Technical Verification

* [ ] **Persistence:** Stopping the application and restarting it preserves all Match and Ticket data.
* [ ] **Compilation:** Code compiles with C++17 standard with zero warnings.
* [ ] **Architecture:** Logic is correctly distributed between Entity classes (Match, Ticket) and Manager classes.