# Offline-First Hockey Stats Tracker Requirements

This document outlines the business and technical requirements for a mobile application designed to track core hockey statistics offline and synchronize them with a central Google Sheet.

**Project Goal:** To enable users to easily record hockey game events (Shots and Penalties) on a mobile device, even without an internet connection, and have this data reliably stored in a Google Sheet for subsequent analysis and decision-making.

**Chosen Architecture:**

- **Client:** Flutter Mobile Application (Android & iOS)
- **Client Local Storage:** Embedded Hive Database within the Flutter App
- **Cloud Backend:** Google Sheets Document
- **Integration Mechanism:** Google Sheets API (accessed via Flutter googleapis package)
- **Core Operational Logic:** Client-side Synchronization Service

**Core System Components:**

1. **FlutterApp (Client):**
   - Runs on the user's mobile device.
   - Provides the user interface for data entry and viewing local stats.
   - Interacts with the LocalDatabase for all data persistence during offline and online states.
   - Contains the SynchronizationService logic.
2. **LocalDatabase (Client Storage):**
   - Uses Hive as the LocalDatabase
   - An embedded database running directly on the mobile device within the FlutterApp.
   - Stores Player, Game, and GameEvent data.
   - Must support efficient querying and reliable data persistence offline.
3. **GoogleSheetsAPI (Integration):**
   - The official API provided by Google for interacting with Google Sheets data.
   - Accessed by the FlutterApp (specifically the SynchronizationService) over the internet.
   - Requires authentication (e.g., Service Account or OAuth).
4. **GoogleSheets (Cloud Storage):**
   - A specific Google Sheet document acting as the central, persistent data store.
   - Structured into multiple sheets (tabs) to hold different types of data (Roster,

Games, GameEvents).

5. **SynchronizationService (Client Logic):**
    - A module within the FlutterApp.
    - Monitors network connectivity.
    - Identifies unsynced data in the LocalDatabase.
    - Formats local data for Google Sheets.
    - Uses GoogleSheetsAPI to push unsynced data to GoogleSheets.
    - Updates sync status in LocalDatabase.
    - Handles API errors, retries, and potential basic conflict resolution.
    - Manages initial loading/sync of Roster and Games from GoogleSheets to LocalDatabase.

**User Flows (Activity-Based Requirements):**

These flows describe the user's interaction with the FlutterApp and the required system behavior, emphasizing offline capability.

- **UF-1: Select Current Game**
    - **Trigger:** User opens the app and needs to select the game they are tracking stats for.
    - **Action:** FlutterApp displays a list of available games.
    - **System Response:** User selects a game. FlutterApp loads the selected game's details and associated roster (from LocalDatabase) into memory for quick access during data entry.
    - **Offline Capability: MUST WORK OFFLINE.** The list of games and associated rosters must be available from the LocalDatabase. Initial game/roster data is loaded into LocalDatabase via UF-6.
- **UF-2: Log a Shot**
    - **Trigger:** A shot event occurs in the game.
    - **Action:** User taps a dedicated "Log Shot" button in the FlutterApp UI. FlutterApp presents input controls for Shot details.
    - **Input Data:**
        - IsGoal: Boolean (Yes/No toggle/checkbox).
        - Team: String (Selector: "Your Team" / "Opponent").
        - ShooterJersey: Number (Selector: Choose from players on the selected team's roster available in LocalDatabase).
        - Assist1Jersey: Number (Optional Selector: Choose from players on "Your Team" roster. Only enabled/relevant if IsGoal is Yes).
        - Assist2Jersey: Number (Optional Selector: Choose from players on "Your Team" roster. Only enabled/relevant if IsGoal is Yes and Assist1Jersey is

        selected).
        - **OnIcePlayers:** Selector: Choose from "Your Team" roster. Only enabled/relevant if IsGoal is Yes..
    - **System Response:** FlutterApp captures input, records current timestamp, associates data with the currently selected game and period. Creates a GameEvent object with EventType="Shot", populating relevant fields (IsGoal, Team, PrimaryPlayerID mapped from ShooterJersey, AssistPlayer1ID, AssistPlayer2ID). Sets IsSynced flag to FALSE. Saves the GameEvent object to the LocalDatabase. Clears input controls for the next entry.
    - **System Response:** FlutterApp captures input, records current timestamp, associates data with the currently selected game and period. Creates a GameEvent object with EventType="Shot", populating relevant fields (IsGoal, Team, PrimaryPlayerID mapped from ShooterJersey, AssistPlayer1ID, AssistPlayer2ID). **If IsGoal is Yes, it captures the list of selected players on ice and stores their IDs in the `onIcePlayerIds` list.** Sets IsSynced flag to FALSE. Saves the GameEvent object to the LocalDatabase. Clears input controls for the next entry.
    - **Offline Capability: MUST WORK OFFLINE.** All input, validation against local roster, timestamping, object creation, and saving to LocalDatabase must function without an internet connection.
- **UF-3: Log a Penalty**
    - **Trigger:** A penalty event occurs in the game.
    - **Action:** User taps a dedicated "Log Penalty" button in the FlutterApp UI. FlutterApp presents input controls for Penalty details.
    - **Input Data:**
        - Team: String (Selector: "Your Team" / "Opponent").
        - PenalizedPlayerJersey: Number (Selector: Choose from players on the selected team's roster available in LocalDatabase).
        - PenaltyType: String (Text Input or Selector from predefined list).
        - PenaltyDuration: Number (Input: Minutes).
    - **System Response:** FlutterApp captures input, records current timestamp, associates data with the currently selected game and period. Creates a GameEvent object with EventType="Penalty", populating relevant fields (Team, PrimaryPlayerID mapped from PenalizedPlayerJersey, PenaltyType, PenaltyDuration). Sets IsSynced flag to FALSE. Saves the GameEvent object to the LocalDatabase. Clears input controls for the next entry.
    - **Offline Capability: MUST WORK OFFLINE.** All input, validation against local roster, timestamping, object creation, and saving to LocalDatabase must function without an internet connection.
- **UF-4: Synchronize Data**
    - **Trigger:** FlutterApp detects network connectivity is available, or user initiates

    sync manually (e.g., via a button).
  - **Action:** SynchronizationService identifies GameEvent records in LocalDatabase where IsSynced is FALSE.
  - **System Response:**
    - For each unsynced GameEvent:
      - SynchronizationService formats the event data into a row structure matching the GoogleSheets GameEvents sheet columns (Timestamp, GameID, Period, EventType, Team, PrimaryPlayerJersey, Assist1Jersey, Assist2Jersey, IsGoal, PenaltyType, PenaltyDuration, etc.). Player/Game IDs from the local database must be translated back to Sheet-compatible identifiers (like Jersey Numbers and Game Date/Opponent, or Sheet row references if applicable).
      - Uses the googleapis package to call the Google Sheets API (specifically, the method to append a row) to add this data to the designated GameEvents sheet in GoogleSheets.
      - **Success:** If the API call is successful, SynchronizationService updates the corresponding GameEvent record in LocalDatabase, setting IsSynced to TRUE and potentially storing the Google Sheet row number for reference.
      - **Failure:** If the API call fails (e.g., network error, API quota), the GameEvent record in LocalDatabase remains unsynced. Implement retry logic with exponential backoff.
    - Provide feedback to the user on sync progress and completion/errors.
  - **Offline Capability: DOES NOT WORK OFFLINE.** Requires an active internet connection to communicate with GoogleSheetsAPI.
- **UF-5: View Local Game Events/Basic Stats**
  - **Trigger:** User wants to review events logged for the current game or view simple stats within the app.
  - **Action:** User navigates to a review or stats screen.
  - **System Response:** FlutterApp queries the LocalDatabase for GameEvent records related to the current game. Displays a list of logged events and/or calculates and displays basic counts (e.g., total shots, total penalties) based *only* on the data currently in the LocalDatabase.
  - **Offline Capability: MUST WORK OFFLINE.** All data displayed comes from the LocalDatabase.
- **UF-6: Initial Setup / Roster & Schedule Sync**
  - **Trigger:** First app launch, or user explicitly requests an update to roster/schedule data.
  - **Action:** User initiates the initial data load/sync.

    - **System Response:** FlutterApp uses googleapis to read data from the Roster and Games sheets in GoogleSheets. Clears existing roster/game data in LocalDatabase (or merges/updates based on a defined strategy). Saves the fetched data into the LocalDatabase.
    - **Offline Capability: DOES NOT WORK OFFLINE.** Requires an active internet connection to read initial data from GoogleSheets.

**Data Model (Entities and Attributes):**

This defines the structure of data to be managed by the system, mirrored between LocalDatabase and GoogleSheets.

- **Entity: Player**
    - id: String (Unique Identifier, e.g., UUID generated by the app or derived from Google Sheet data). Primary Key in LocalDatabase.
    - jerseyNumber: Integer (Required).
    - name: String (Required).
    - teamId: String (Link to Team Entity, if multiple teams are supported).
    - googleSheetRow: Integer (Optional, reference to the row in the GoogleSheets 'Roster' sheet for sync tracking).
- **Entity: Game**
    - id: String (Unique Identifier, e.g., UUID or derived from Google Sheet data). Primary Key in LocalDatabase.
    - date: DateTime (Required).
    - opponent: String (Required).
    - location: String (Optional).
    - googleSheetRow: Integer (Optional, reference to the row in the GoogleSheets 'Games' sheet for sync tracking).
- **Entity: GameEvent**
    - id: String (Unique Identifier, generated locally by the FlutterApp). Primary Key in LocalDatabase.
    - gameId: String (Required, Foreign Key linking to Game Entity).
    - timestamp: DateTime (Required, when the event was logged on the device).
    - period: Integer (Required, the game period).
    - eventType: String (Required, "Shot" or "Penalty").
    - team: String (Required, "Your Team" or "Opponent").
    - primaryPlayerId: String (Required, Foreign Key linking to Player Entity - the shooter or penalized player).
    - assistPlayer1Id: String (Optional, Foreign Key linking to Player Entity - Assist 1).
    - assistPlayer2Id: String (Optional, Foreign Key linking to Player Entity - Assist

2).
    - isGoal: Boolean (Optional, relevant only if eventType is "Shot").
    - penaltyType: String (Optional, relevant only if eventType is "Penalty").
    - penaltyDuration: Integer (Optional, relevant only if eventType is "Penalty").
    - isSynced: Boolean (Required, Default: false). Flag for SynchronizationService.
    - googleSheetRow: Integer (Optional, reference to the row in the GoogleSheets 'GameEvents' sheet after successful sync).

*(Note: A Team entity could be added if tracking multiple teams, linking Players and Games to specific teams.)*

**Technical Specifications:**

- **Flutter Development:** Utilize Flutter SDK for building the cross-platform mobile application.
- **Local Database Integration:** Choose and integrate a suitable Flutter-compatible local database (sqflite, Isar, Hive). Define the schema based on the Data Model. Implement standard database operations (CRUD, querying).
- **Google Sheets API Integration:**
    - Add the googleapis package to the Flutter project.
    - Configure Google Cloud Project: Enable Google Sheets API. Create a Service Account or set up OAuth consent screen and credentials. Securely manage credentials within the app (e.g., using environment variables or secure storage).
    - Implement API calls using the googleapis library for:
        - Reading rows from 'Roster' and 'Games' sheets (UF-6).
        - Appending rows to the 'GameEvents' sheet (UF-4).
        - Potentially updating cells (e.g., marking a game as completed - future).
- **Synchronization Service Implementation:**
    - Implement network connectivity checking (e.g., using connectivity_plus package).
    - Implement background or foreground logic to trigger sync when online.
    - Query LocalDatabase for unsynced GameEvent records (isSynced = false).
    - Map local GameEvent data (including resolving Player/Game IDs back to Jersey Numbers/Game identifiers for the Sheet) to the column structure of the GoogleSheets 'GameEvents' sheet.
    - Use batching for API writes if possible to improve efficiency for multiple unsynced records.
    - Implement error handling for API calls (network issues, authentication errors, API errors).

    - Implement retry logic for failed sync attempts.
    - Update isSynced status in LocalDatabase upon successful write.
    - Implement initial sync logic for Roster/Games data (UF-6).
- **User Interface:** Design a clean, intuitive UI in Flutter with large, easily tappable buttons for the core activities ("Log Shot", "Log Penalty"). Use appropriate Flutter widgets for data input (checkboxes, dropdowns, text fields). Ensure responsiveness across different mobile screen sizes.
- **State Management:** Use a suitable Flutter state management solution (e.g., Provider, Riverpod, Bloc, GetX) to manage application state, including game context, unsynced data status, and UI updates.

**Non-Functional Requirements:**

- **Reliability:** Ensure data is durably stored in the LocalDatabase immediately upon entry, even if the app is closed or crashes. The sync process must guarantee data integrity and minimize data loss.
- **Performance:** Data entry and local data retrieval should be fast and smooth, even with a growing number of local events. The sync process should ideally not block the main UI thread.
- **Usability:** The activity-based data entry UI must be extremely easy and quick to use in a game environment.
- **Security:** Google API credentials must be stored and used securely. Data transmitted to Google Sheets should be protected by standard Google security measures.
- **Scalability:** The LocalDatabase should handle a reasonable number of games/events before performance degrades. GoogleSheets has its own limitations on row count and API request quotas, which should be considered for very high-volume usage.