Software Requirements Specification

Legacy System: HockeyStatsTS (v1.0)

Detailed System Audit

January 11, 2026

Contents

Chapter 1

System Overview

1.1 Product Scope

HockeyStatsTS is a React-based Single Page Application (SPA) designed for post-game statistical logging. It allows administrators to define teams and players, configure game metadata, and log events (goals, shots, penalties) by interacting with a digital rink representation. The system relies on Firebase for backend services and uses a custom Object-Oriented Programming (OOP) layer within the frontend for business logic.

1.2 User Roles

- **Public User:** Can view the Landing Page (Login).
- **Administrator:** Identified via a hardcoded `adminUids` array in the source code. Has full access to CRUD operations and Game Logging.

Chapter 2

Detailed Functional Requirements

2.1 Module 1: Authentication & Routing

File Reference: `App.tsx`, `AuthPage.tsx`

2.1.1 Authentication Flow

1. **Initialization:** On application load, `App.tsx` triggers a Firebase `onAuthStateChanged` listener.
2. **Verification:**
   - If a user object exists, the system checks if `user.uid` is present in the `adminUids` array.
   - State variable `isSignedIn` is set to `true` ONLY if the UID matches.
3. **Routing Guard:**
   - If `isSignedIn` is false, any attempt to access protected routes (starts with `/admin`, `/game`, etc.) triggers a redirect to `/`.
   - A `LoadingSpinner` is displayed while the auth check resolves.

2.1.2 Route Definitions

The application uses `createBrowserRouter` with the following structure:

Path	Component	Functionality
`/`	`HomePage`	Dashboard showing menu options based on auth status.
`/start`	`StartPage`	**Loader:** Fetches all Teams. Displays Game Setup form.
`/game`	`GamePage`	The core logging interface. Requires `location.state` with setup data.
`/saved-games`	`SavedGamesPage`	Lists historical games from Firestore.
`/handle-teams`	`TeamCRUDPage`	**Loader:** Fetches Teams + Games. Lists teams with filters.
`/handle-teams/create`	`CreateTeamPage`	Form to add new Team entities.
`/handle-players`	`PlayerCRUDPage`	**Loader:** Fetches Players + Teams. Lists players.

2.2 Module 2: Master Data Management

File Reference: `TeamCRUDPage.tsx`, `PlayerCRUDPage.tsx`

2.2.1 Team Management

1. **Data Loading:** The route loader fetches all teams via `TeamService.getAllTeams()` and games via `GameService.getAllGames()` (for dependency checking).
2. **Filtering:** Users can filter the list by Name (text input), Season (dropdown), and Championship (dropdown).
3. **Deletion Logic:**
   - User clicks Delete.
   - System prompts `window.confirm`.
   - Calls `TeamService.deleteTeam(id)`.
   - **Constraint:** Does not check if the team has existing games before deletion (potential for orphaned data).

2.3 Module 3: The Game Logging Engine

File Reference: `GamePage.tsx`

2.3.1 State Management

The page maintains complex local state:

- `gameSetup`: Metadata passed from StartPage.
- `actions`: Array of `IGameAction` representing the event log.
- `homeScore / awayScore`: Computed local state objects.
- `modalStep`: Finite State Machine (`'action'` → `'player'` → `'assist'` → `'confirm'`).

2.3.2 Event Creation Workflow

1. **Trigger:** User clicks `InteractiveRink`.
2. **Step 1: Action Selection (Modal A)**
   - **UI:** Displays Time Input (Period, Min, Sec).
   - **Grid:** Two distinct rows. Row 1 = Home Team Actions. Row 2 = Away Team Actions.
   - **Visuals:** Icons use the respective Team's Primary/Secondary colors passed via props.
   - **Selection:** User clicks an Action Type (e.g., GOAL). System stores `currentAction.type` and `currentAction.team`.
3. **Step 2: Player Selection (Modal B)**
   - **UI:** Lists players from the selected team's roster.
   - **Selection:** User selects a player.
   - **Branching:** If Action == GOAL, proceed to Step 3. Else, proceed to Step 4.

4. **Step 3: Assist Selection (Modal C)**
    - **UI:** Multi-select list of teammates (excluding the scorer).
    - **Actions:** User selects 0, 1, or 2 players and clicks "Next".
5. **Step 4: Confirmation (Modal D)**
    - **UI:** Displays summary card (Action, Time, Player, Location).
    - **Controls:** "Confirm" (Commit), "Edit" (Step Back), "Cancel" (Abort).
6. **Commit:**
    - New action pushed to `actions` array.
    - `updateScores` function recalculates the scoreboard immediately.
    - `autosave` triggers `localStorage.setItem('unfinishedGame')`.

2.3.3 Autosave & Persistence

- The system writes the entire `gameState` object to `localStorage` under the key `'unfinishedGame'` on every action change if autosave is toggled on.
- On `GamePage` mount, it checks for this key and hydrates state if found.