# Software Requirements Specification (SRS) - Online Buzzer System

**Version:** 1.0 **Date:** October 26, 2023 **Author:** AI Assistant

## 1. Introduction

### 1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to precisely define the functional and non-functional requirements for the "Online Buzzer System." This web-based application is designed to provide a fair, reliable, and accessible real-time buzzer mechanism for quiz-style games and interactive events. It aims to eliminate issues related to simultaneous button presses and delayed notifications, ensuring a clear "first-in" determination. This document serves as a foundational agreement for development, testing, and deployment.

### 1.2 Scope

The Online Buzzer System will encompass two primary user interfaces: a Host/Controller View and a Participant/Player View. Its core functionality will center around managing distinct "Rooms" for individual event sessions, supporting multiple participants per Room. The system prioritizes real-time performance, cross-platform compatibility (desktop and mobile web browsers), ease of deployment, and intuitive usability. It will rely on external cloud services for real-time data synchronization, thus minimizing server-side management requirements.

### 1.3 Definitions, Acronyms, and Abbreviations

| Term | Definition |
|---|---|
| **Buzzer State** | The current operational status of the buzzer within a Room (e.g., "Armed," "Locked"). |
| **Firebase** | Google's Backend-as-a-Service (BaaS) platform, providing real-time database capabilities. |
| **GitHub Pages** | A static site hosting service provided by GitHub. |
| **Host** | The user (quizmaster, moderator) responsible for controlling the game session. |
| **Latency** | The delay between an action and its visible effect across the system. |

| Term | Definition |
|---|---|
| **Participant** | An individual or team member utilizing the buzzer to respond. |
| **Real-time** | Data synchronization and updates occurring instantaneously across all connected clients. |
| **Room** | A unique, isolated instance of a game session. |
| **SRS** | Software Requirements Specification. |
| **UI** | User Interface. |
| **UX** | User Experience. |

### 1.4 References

- Firebase Realtime Database Documentation: `https://firebase.google.com/docs/database`
- GitHub Pages Documentation: `https://docs.github.com/en/pages`
- W3C Web Standards for HTML, CSS, JavaScript.

## 2. Overall Description

### 2.1 Product Perspective

The Online Buzzer System is a standalone web application designed for zero-backend-management deployment. It integrates with cloud-based real-time database services (e.g., Firebase) to manage and synchronize game state across all connected clients. This eliminates the need for users to provision, configure, or maintain a traditional server infrastructure.

### 2.2 Product Features

The system will provide the following key features:

- **Real-time Buzzer Control:** Host-driven arming and resetting of the buzzer.
- **First-In-First-Served Logic:** Guaranteed determination of the first participant to buzz.
- **Cross-Platform Accessibility:** Full functionality across desktop and mobile web browsers.
- **Unique Room Sessions:** Isolation of multiple simultaneous events.
- **Intuitive User Interfaces:** Separate, optimized UIs for Hosts and Participants.

## 2.3 User Classes and Characteristics

### 2.3.1 Host User

- **Description:** Typically an event organizer or quizmaster. Requires control over the game flow and clear visibility of participant responses.
- **Characteristics:** Possesses administrative control within a Room. Needs to initiate and reset rounds.
- **Environment:** Primarily uses a laptop or desktop computer with a stable internet connection.

### 2.3.2 Participant User

- **Description:** An individual or team member competing in the event. Requires a simple, responsive interface to "buzz in."
- **Characteristics:** Focuses on quick reaction time. Needs clear feedback on buzzer status and success/failure.
- **Environment:** Primarily uses mobile phones or tablets, but could also use laptops/desktops, requiring a stable internet connection.

## 2.4 Operating Environment

- **Client-side:** Any modern web browser supporting HTML5, CSS3, and ECMAScript 2015+ (e.g., Chrome, Firefox, Safari, Edge, Brave, Opera) on various operating systems (Windows, macOS, Linux, iOS, Android).
- **Backend Services:** Google Firebase Realtime Database.
- **Hosting Platform:** GitHub Pages.
- **Connectivity:** Requires a reliable internet connection for all users.

## 2.5 Design and Implementation Constraints

- **Technology Stack:** Must utilize HTML, CSS, and JavaScript for the front-end. Firebase Realtime Database must be used for real-time backend synchronization.
- **Deployment:** Must be deployable as a static site, compatible with GitHub Pages.
- **Serverless Backend:** No custom server-side code or infrastructure to be managed by the end-user.
- **Third-Party Services:** Reliance on Firebase for database and real-time functions.

## 3. Functional Requirements

## 3.1 Room Management (FR-RM)

### FR-RM-001: Generate Unique Room ID

- **Description:** The system shall automatically generate a unique, short, alphanumeric identifier (Room ID) when a Host accesses the Host View without a Room ID specified in the URL.
- **Priority:** High
- **Source:** Host User

#### FR-RM-002: Display Room ID

- **Description:** The system shall clearly display the current Room ID in both the Host and Participant Views.
- **Priority:** High
- **Source:** Host User, Participant User

#### FR-RM-003: Generate Participant Link

- **Description:** The system shall provide a shareable URL for the Participant View, automatically embedding the current Room ID, to allow participants to easily join the specific session.
- **Priority:** High
- **Source:** Host User

#### FR-RM-004: Join Existing Room

- **Description:** Participants shall be able to join an existing Room by navigating to the Participant View URL containing a valid Room ID.
- **Priority:** High
- **Source:** Participant User

#### FR-RM-005: Room Persistence

- **Description:** A Room's state (buzzer status, winner) shall persist as long as at least one user (Host or Participant) is actively connected or until explicitly reset by the Host.
- **Priority:** Medium
- **Source:** Host User

### 3.2 Host View Features (FR-HOST)

#### FR-HOST-001: Display Buzzer State

- **Description:** The Host View shall clearly display the current Buzzer State ("Armed" or "Locked") to the Host.
- **Priority:** High
- **Source:** Host User

#### FR-HOST-002: Display Winner

- **Description:** When the Buzzer State is "Locked" by a Participant's buzz, the Host View shall instantly display the name of the first Participant who buzzed in.
- **Priority:** High
- **Source:** Host User

#### FR-HOST-003: Reset Buzzer Functionality

- **Description:** The Host View shall provide a prominent "Reset Buzzer" button.
- **Priority:** High

- **Source:** Host User

### FR-HOST-004: Reset Buzzer Action

- **Description:** Upon activation, the "Reset Buzzer" button shall reset the Buzzer State to "Armed" and clear any previously recorded winner, making the buzzer available for the next question.
- **Priority:** High
- **Source:** Host User

### FR-HOST-005: Visual Feedback for Host

- **Description:** The Host View shall provide clear visual cues (e.g., color changes, status messages) to reflect changes in Buzzer State and Winner display.
- **Priority:** High
- **Source:** Host User

## 3.3 Participant View Features (FR-PLAYER)

### FR-PLAYER-001: Team Name Input

- **Description:** Upon joining a Room, the Participant View shall prompt the user to enter their team/individual name before enabling the buzzer functionality.
- **Priority:** High
- **Source:** Participant User

### FR-PLAYER-002: Persist Team Name

- **Description:** The system shall store the entered team/player name in the browser's local storage for that specific Room, pre-filling it on subsequent visits or refreshes.
- **Priority:** Medium
- **Source:** Participant User

### FR-PLAYER-003: Buzzer Button Display

- **Description:** The Participant View shall present a large, distinct, and easily tappable/clickable "Buzzer Button."
- **Priority:** High
- **Source:** Participant User

### FR-PLAYER-004: Buzzer Button Enablement

- **Description:** The Buzzer Button shall be interactively enabled only when the Buzzer State is "Armed" by the Host.
- **Priority:** High
- **Source:** Participant User

### FR-PLAYER-005: Successful Buzz Feedback

- **Description:** Upon a successful buzz (being the first to press when "Armed"), the Buzzer Button shall immediately become disabled and provide prominent visual feedback (e.g., animated glow, distinct color) indicating the participant's success.
- **Priority:** High
- **Source:** Participant User

**FR-PLAYER-006: Unsuccessful Buzz Feedback**

- **Description:** If another participant buzzes in first, the Buzzer Button shall become disabled, and the Participant View shall display a message indicating who successfully buzzed first.
- **Priority:** High
- **Source:** Participant User

**FR-PLAYER-007: General Buzzer State Feedback**

- **Description:** The Participant View shall display clear textual status messages reflecting the current Buzzer State (e.g., "Buzzers Armed!", "Buzzers Locked - Waiting for Host Reset").
- **Priority:** High
- **Source:** Participant User

### 3.4 Real-time Communication and Core Logic (FR-RT)

**FR-RT-001: Instant State Synchronization**

- **Description:** All changes to the Buzzer State (Armed/Locked) and Winner information shall be synchronized across all connected Host and Participant Views in real-time.
- **Priority:** Critical
- **Source:** All Users

**FR-RT-002: First-In-First-Served Arbitration**

- **Description:** The system shall implement a robust "first-in-first-served" mechanism using database transactions, guaranteeing that only the *absolute first* valid buzz during an "Armed" state is registered as the winner. All other near-simultaneous buzzes will be rejected.
- **Priority:** Critical
- **Source:** All Users

**FR-RT-003: Prevent Buzz when Locked**

- **Description:** The system shall technically prevent any Participant from successfully registering a buzz if the Buzzer State is already "Locked" (even if their local button UI hasn't fully updated yet).
- **Priority:** High
- **Source:** Participant User

## 4. Non-Functional Requirements

### 4.1 Performance (NFR-PERF)

**NFR-PERF-001: Low Latency**

- **Description:** The end-to-end latency for a buzz event (from button press to winner display on all screens) shall average less than 100 milliseconds under typical network conditions.
- **Priority:** Critical

**NFR-PERF-002: Scalability**

- **Description:** The system shall efficiently support up to 100 concurrent participants within a single Room without significant degradation in real-time performance.
- **Priority:** High

**NFR-PERF-003: UI Responsiveness**

- **Description:** Both Host and Participant UIs shall be highly responsive, with fast loading times and smooth transitions across various devices and screen sizes.
- **Priority:** High

### 4.2 Security (NFR-SEC)

**NFR-SEC-001: Data Integrity**

- **Description:** Firebase security rules shall be configured to prevent unauthorized or malicious alteration of the Buzzer State or Winner information by Participant users. Only the Host (or the "first-in" transaction) should be able to modify the core buzzer data.
- **Priority:** High

**NFR-SEC-002: Data Confidentiality**

- **Description:** The system shall not collect or store any personally identifiable information (PII) beyond the team/player name provided by participants for the duration of the event session.
- **Priority:** High

### 4.3 Usability (NFR-USABILITY)

**NFR-USABILITY-001: Intuitive Interface**

- **Description:** The UI for both Host and Participants shall be clean, simple, and self-explanatory, requiring minimal instructions for use.
- **Priority:** High

**NFR-USABILITY-002: Minimal Setup Effort**

- **Description:** Deployment and initial setup for the Host shall involve only uploading files to GitHub Pages and configuring Firebase API keys/rules. No complex server-side setup or database management is required.
- **Priority:** High

**NFR-USABILITY-003: Cross-Device Experience**

- **Description:** The system shall provide an optimized and consistent user experience across desktop, tablet, and mobile browsers, adapting gracefully to different screen sizes and orientations.
- **Priority:** High

## 4.4 Maintainability (NFR-MAINT)

### NFR-MAINT-001: Modular Codebase

- **Description:** The JavaScript code shall be modular, with clear separation of concerns (e.g., Firebase logic, UI updates, event handlers) to facilitate future updates and bug fixes.
- **Priority:** Medium

### NFR-MAINT-002: Well-Commented Code

- **Description:** The codebase (HTML, CSS, JavaScript) shall be adequately commented to explain complex logic and design choices.
- **Priority:** Medium

### NFR-MAINT-003: External Service Reliance

- **Description:** The system's reliance on managed cloud services (Firebase, GitHub Pages) shall minimize long-term maintenance overhead for the end-user deployer.
- **Priority:** High

## 5. Appendices

### 5.1 Proposed Technology Stack

- **Frontend:** HTML, CSS, Vanilla JavaScript.
- **Real-time Backend:** Google Firebase Realtime Database (client-side SDK integration).
- **Hosting:** GitHub Pages.