Software Requirements Specification (SRS) - Online Buzzer System

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for a web-based, real-time online buzzer system designed for quiz and event facilitators. This system aims to provide a fair, reliable, and accessible method for participants to "buzz in" for questions, addressing common issues faced in live events such as latency and simultaneous presses.

1.2 Scope

The system will consist of two primary interfaces: a Host/Quizmaster View and a Participant/Player View. It will support multiple participants (teams or individuals) in a designated "room" for a single event session. The system will prioritize real-time responsiveness, cross-platform compatibility (desktop and mobile), and ease of deployment/use.

1.3 Definitions & Acronyms

- **AI Odyssey:** The example event for which this system is designed.
- **Host:** The quizmaster or event organizer managing the buzzer system.
- **Participant:** An individual or team using the buzzer system.
- **Room:** A unique session instance for a single quiz/event.
- **Buzzer State:** The current operational status of the buzzer system (e.g., Armed, Locked).
- **Real-time:** Updates reflected across all connected clients with minimal delay.
- **Firebase:** Google's Backend-as-a-Service (BaaS) platform, including Realtime Database.
- **GitHub Pages:** A static site hosting service offered by GitHub.
- **SRS:** Software Requirements Specification.

2. Overall Description

2.1 Product Perspective

The Online Buzzer System is a standalone web application. It integrates with cloud-based real-time database services to manage shared state across all connected users without requiring a dedicated server to be managed by the user.

2.2 User Classes and Characteristics

- **Host/Quizmaster:**
  - **Characteristics:** Needs clear display of buzzer status and winner, ability to arm/reset buzzers.
  - **Requirements:** Desktop/laptop access, stable internet, ability to share player link.
- **Participants/Teams:**

- **Characteristics:** Needs a simple, large, responsive buzzer button. Needs clear feedback on buzzer state and if their buzz was successful.
- **Requirements:** Mobile phone, tablet, or laptop access, stable internet, ability to input team name.

2.3 Operating Environment

- **Client-side:** Any modern web browser (Chrome, Firefox, Safari, Edge) on desktop, tablet, or mobile operating systems (iOS, Android, Windows, macOS, Linux).
- **Backend:** Firebase Realtime Database (managed by Google).
- **Hosting:** GitHub Pages (managed by GitHub).

3. System Features

3.1 Host/Quizmaster View Features

- **Display Room ID:** Clearly show the unique identifier for the current game session.
- **Player Link Generation:** Provide an easy-to-copy link for participants to join the specific room.
- **Buzzer State Display:** Clearly indicate if buzzers are "Armed" (active) or "Locked" (inactive).
- **Winner Display:** Instantly show the name of the first participant to buzz in when buzzers are locked.
- **Reset Functionality:** A prominent button to reset the buzzer state, clearing the winner and re-arming buzzers for the next question.
- **Visual Feedback:** Color changes or animations to quickly convey state changes (e.g., red for locked, green for armed).

3.2 Participant/Player View Features

- **Room Joining:** Allow participants to enter a room via a unique URL (e.g., `player.html?room=ROOM_ID`).
- **Team Name Input:** Prompt participants to enter their team/individual name before activating the buzzer.
- **Large Buzzer Button:** A prominent, easy-to-tap/click button for buzzing in.
- **Buzzer State Feedback:**
  - Button enabled/disabled based on "Armed"/"Locked" state.
  - Visual/textual message indicating if buzzers are active, locked, or if someone else has buzzed.
  - Clear indication if *their* buzz was the winning buzz.
- **Persistence:** Remember the team name in local storage for convenience across page refreshes.

3.3 Real-time Communication

- **Instant Updates:** All changes to the buzzer state (armed, locked, winner) must be reflected across all connected Host and Participant views in real-time.
- **First-In-First-Served Logic:** The system must guarantee that only the *absolute first* participant to press the button when buzzers are armed is registered as the winner,

regardless of near-simultaneous presses.

4. Non-Functional Requirements

4.1 Performance

- **Latency:** Updates to buzzer state and winner display must occur within milliseconds (ideally <100ms) for fair play, especially for buzz-in speed. Firebase Realtime Database is designed for this.
- **Scalability:** The system should support up to 50-100 concurrent participants without significant degradation in performance (within typical Firebase free tier limits).
- **Responsiveness:** The UI should be highly responsive on various screen sizes and orientations (mobile-first design).

4.2 Security

- **Data Integrity:** Firebase security rules must be configured to prevent unauthorized tampering with buzzer state (e.g., players resetting buzzers or falsely declaring themselves winners).
- **Confidentiality:** No sensitive user data will be collected or stored beyond temporary team names for the duration of a session.

4.3 Usability

- **Intuitive UI:** Interfaces for both Host and Participants must be simple, clean, and self-explanatory.
- **Minimal Setup:** Host setup should only involve sharing a URL. Participant setup should only involve entering a name.

4.4 Portability

- **Browser Agnostic:** Compatible with all major modern web browsers.
- **Platform Agnostic:** Fully functional on desktop, laptop, tablet, and mobile devices.

4.5 Maintainability

- The codebase will be modular and well-commented to facilitate future updates or bug fixes.
- Leverage managed cloud services (Firebase, GitHub Pages) to minimize maintenance overhead for the deployer.

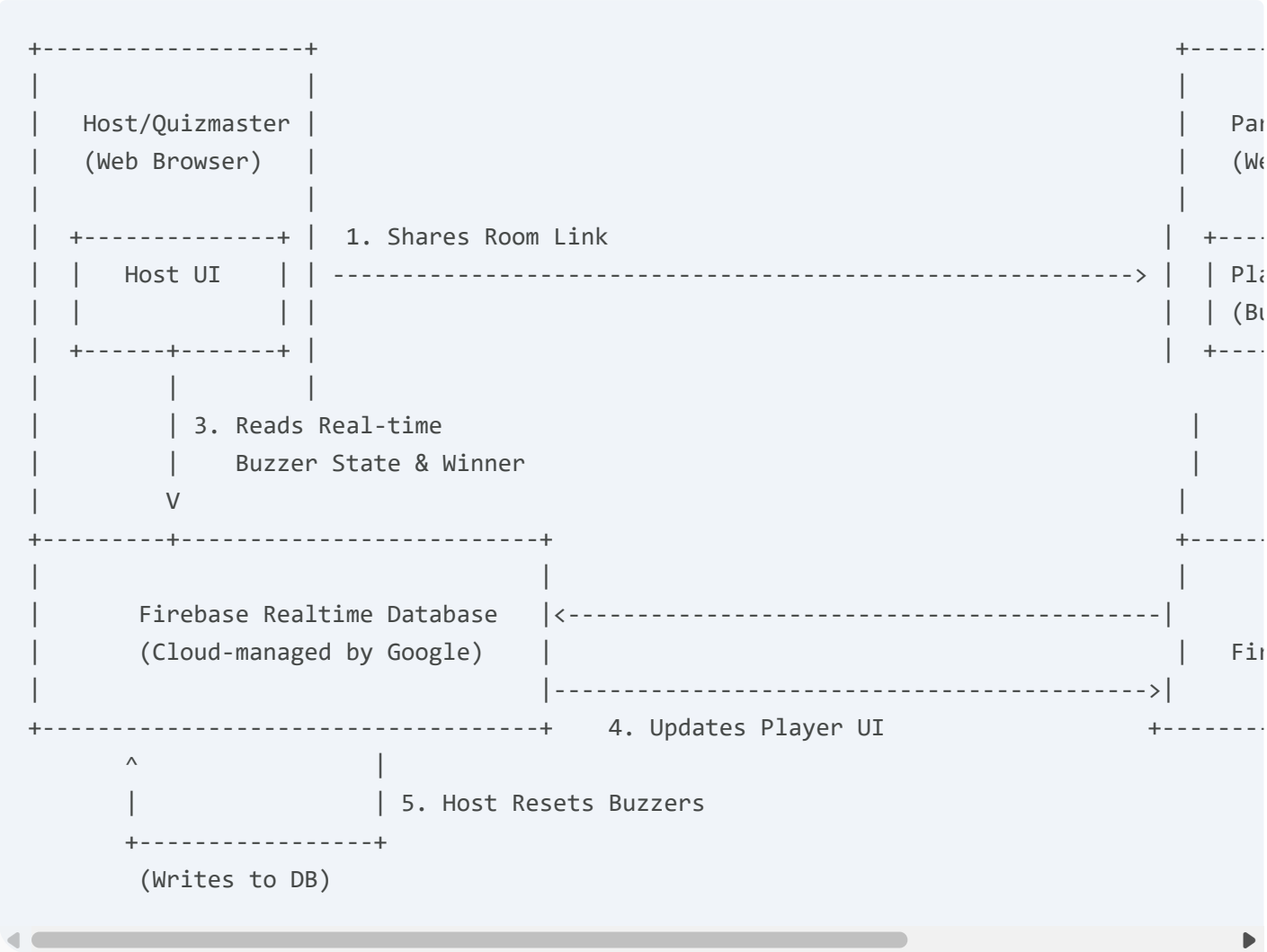
5. Proposed Technology Stack & Overview

Based on the requirements, especially the need for real-time, ease of deployment, and cross-platform access without managing a server, the following stack is highly recommended:

- **Frontend:** HTML, CSS, JavaScript (standard web technologies)
  - **Libraries:** No heavy frameworks needed; vanilla JS for simplicity.
- **Real-time Backend: Google Firebase Realtime Database**
  - **Why:** Provides a NoSQL cloud database that synchronizes data across all connected clients in real-time. Its "transactions" feature is perfect for implementing the critical "first-in-first-served" logic robustly. It abstracts away server management, WebSockets, and complex scaling.

- **Hosting: GitHub Pages**
  - **Why:** Free, incredibly easy to deploy static web applications (HTML, CSS, JS). As Firebase handles the "dynamic" parts, GitHub Pages is a perfect fit.

System Overview Diagram



Key Advantages of this Stack:

- **Zero Server Management:** You never have to worry about provisioning or maintaining a server.
- **Instant Global Deployment:** Once pushed to GitHub, it's live. Firebase is globally distributed.
- **Mobile-First by Design:** Standard web technologies ensure it works on any device.
- **Robust Real-time:** Firebase is built for this exact use case, ensuring fair and quick buzz-ins.
- **Free for Most Events:** Both Firebase and GitHub Pages offer very generous free tiers, easily covering a typical event's usage.