Software Design Document: The Gaming Room

# 1. Introduction

The Gaming Room project is a software design initiative for a client seeking to develop a cross-platform game application. The client, The Gaming Room, aims to deliver a consistent and engaging gaming experience across Android, iOS, Windows, and web platforms. This document outlines the software design, architecture, and implementation strategies for the application.

# 2. System Overview

The system is a multiplayer trivia game that allows users to join game rooms, answer questions, and compete for high scores. The application will be hosted on a centralized server and accessed via platform-specific clients. The server will manage game state, user sessions, and scoring logic.

# 3. Requirements

## 3.1 Functional Requirements

- Users can create and join game rooms.

- The server manages game state and synchronizes player actions.

- Players receive trivia questions and submit answers.

- Scores are calculated and displayed in real-time.

## 3.2 Non-Functional Requirements

- The system must support at least 100 concurrent users.

- The application must be responsive and platform-independent.

- Data must be securely transmitted and stored.

# 4. Architecture

The system follows a client-server architecture. The server is implemented using a RESTful API and WebSocket for real-time communication. Clients are developed using platform-specific frameworks (e.g., React Native for mobile, React for web).

# 5. Component Descriptions

- Game Server: Handles game logic, user sessions, and data storage.

- Client Application: Provides user interface and communicates with the server.

- Database: Stores user data, game history, and question bank.

# 6. Data Models

User: { id, username, email, score }

GameRoom: { id, name, players, currentQuestion }

Question: { id, text, options, correctAnswer }

# 7. User Interface Design

The UI includes screens for login, game room selection, question display, and scoreboards. The design emphasizes usability and responsiveness across devices.

# 8. Technology Stack

- Frontend: React, React Native

- Backend: Node.js, Express

- Database: MongoDB

- Real-time Communication: WebSocket

# 9. Security Considerations

- Use HTTPS for all communications.

- Implement authentication and authorization.

- Sanitize inputs to prevent injection attacks.

# 10. Testing Strategy

- Unit testing for individual components.

- Integration testing for client-server interactions.

- User acceptance testing with real users.

# 11. Future Enhancements

- Add support for additional game types.

- Implement AI-based question generation.

- Introduce social features like friend lists and chat.