<The Gaming Room>

# Software Design Template

Version 3.0

## Table of Contents

[Software Design Template 1](#_Toc21086455)

[Table of Contents 2](#_Toc21086456)

[Document Revision History 2](#_Toc21086457)

[Executive Summary 3](#_Toc21086458)

[Requirements 3](#_Toc21086459)

[Design Constraints 3](#_Toc21086459)

[Rationale 3](#_Toc21086460)

## Document Revision History

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/18/2024 | <Austin Mejias> | <Initial draft of software design> |
| 2.0 | 08/04/2024 | <Austin Mejias> | <Initial draft of software design> |
| 3.0 | 08/18/2024 | <Austin Mejias> | <Added Recommendations> |

**Executive Summary**

The Gaming Room, a client of Creative Technology Solutions, wants to make a web version of their Android game, "Draw It or Lose It." This game is like the game "Pictionary" where teams guess what is being drawn. They need the game to work on multiple platforms and ensure that game and team names are unique.

**Requirements**

The Gaming Room has several key needs for their new game:

1. **Business Needs:**
   * Create a web-based game that works on multiple platforms.
2. **Technical Needs:**
   * The game must support multiple teams, with multiple players in each team.
   * Only one game should run at a time.
   * One game round should last 30-seconds
   * A steal should last 15-seconds.
   * Create unique IDs for each game, team, and player.

**Design Constraints**

1. **One Game Instance:**
   * Only one instance of the GameService class should be allowed in memory at any time, using the singleton pattern.
2. **Unique Names:**
   * Ensure each game and team name is unique to avoid duplication.
3. **Unique IDs:**
   * Generate unique IDs for each game, team, and player.
4. In Game Timer
   * A 30-second timer should count off and the picture should finish rendering by the end of the round. If the team guesses incorrectly the team trying to steal should have 15-seconds to guess.

**Rationale**

1. **One Game Service Instance:**
   * **Why:** Using the singleton pattern ensures that only one game service instance exists, making game management easier and more consistent.
2. **Unique Names:**
   * **Why:** Ensuring unique names prevents confusion and helps users easily check if a name is available.
3. **Unique IDs:**
   * **Why:** Generating unique IDs for games, teams, and players helps keep track of them and prevents mix-ups.

**Recommendations**

1**. Operating Platform**

* **Recommendation:** Use Amazon Web Services (AWS) for hosting the game.
  + AWS can handle the game running on multiple devices.
  + It’s reliable, fast, and secure, with tools to help manage the game easily.

2**. Operating Systems Architectures**

* **Recommendation:** Use microservices architecture on AWS.
  + Breaks the game into smaller parts that work independently.
  + Easier to update and fix and won’t crash the whole system if one part has a problem.

3**. Storage Management**

* **Recommendation:** Use Amazon S3 for storing game data and Amazon RDS for handling user data.
  + Amazon S3 keeps game files safe and easy to access.
  + Amazon RDS manages user information without needing much setup or maintenance.

4**. Memory Management**

* **Recommendation:** Run the game on AWS EC2 instances with tools like Amazon ElastiCache.
  + Handles memory automatically so the game runs smoothly.
  + Can adjust to more players or bigger data needs without slowing down.

5**. Distributed Systems and Networks**

* **Recommendation:** Use RESTful APIs for communication and AWS tools like SQS and SNS for messaging.
  + Allows different parts of the game to talk to each other, no matter the device.
  + AWS ensures the game stays connected and responsive, even with lots of players**.**

6**. Security**

* **Recommendation:** Protect user data with AWS’s security features, like KMS, SSL/TLS, Cognito, IAM, and WAF.
  + Encrypts data to keep it safe.
  + Manages user logins securely.
  + Protects the game from attacks and ensures it stays up and running.