

Creative Technology Solutions

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/19/2025 | Valerie Dawson | Initial draft. Includes software design constraints, UML analysis and object-oriented principles. |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)/

The Gaming Room seeks to expand its existing Android-based game. Draw It or Lose It, into a web-based application that can serve multiple platforms. The game, inspired by Win, Lose, or Draw, involves teams competing to guess an image rendered from a library of stock drawings. This document outlines a structured approach to the software design, ensuring scalability, maintainability, and optimal performance while meeting client requirements. The proposed solution includes implementing a centralized game service using the singleton pattern, ensuring unique identifiers for games, teams, and players, and enforcing name uniqueness.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

**Web-Based Distributed Environment**: The application must support multiple platforms while maintaining consistency and synchronization across instances.

**Singleton Pattern for Game Management**: To ensure only one instance of the game exists in memory at a time, reducing redundancy and optimizing.

**Unique Identifiers**: Each game, team, and player must have a distinct identifier to prevent duplication and ensure proper data management.

**Unique Names for Games and Teams**: Implementing name validation mechanism to prevent duplicate names.

**Scalability and Performance:** The application should handle multiple teams and players efficiently while minimizing latency.

**Security Considerations**: The application should protect user data and game state integrity.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

**GameService Class** (Singleton Pattern)

* Manages game instances and ensures only one instance exists at a time
* Holds a list of Game objects
* Provides methods to add and retrieve games

**Game Class** (Inherits from Entity)

* Represents a game instance with a unique ID and name.
* Implements methods to retrieve its attributes.

**Entity Class** (Base Class)

* Common attributes for all entities (id and name)
* Inherited by Game, Team and Player classes.

**Team Class** (Inherits from Entity)

* Represents a team with a unique ID and name,
* Manages a list of Player objects.

**Player Class** (Inherits from Entity)

* Represents an individual player with a unique ID and name.

The UML model follows key object-oriented principles:

Encapsulation: Each class maintains private attributes with public getter methods.

Inheritance: The Entity class allows Game, Team, and Player to share common attributes and behaviors.

Singleton Pattern: Ensures only one instance of GameService exists, maintaining game state consistency.

Iterator Pattern: Used in addGame() and get getGame() methods to manage game instances efficiently.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers provide stability and security but are less common for hosting web applications. | Linux is widely used for web hosting due to its flexibility, performance, and cost-effectiveness. | Windows servers offer seamless integration with Microsoft technologies but may require additional licensing costs. | Mobile devices are not typically used for hosting but must communicate efficiently with the server. |
| **Client Side** | Development for macOS requires Xcode and Apple’s guidelines compliance. | Linux desktop applications are less common, but web-based access ensures cross-platform usability. | Windows provides broad compatibility, but development may require additional testing for various versions. | Mobile applications need responsive design and optimization for touch interfaces. |
| **Development Tools** | Swift and Objective-C for native applications; cross-platform tools like React. | Java, Python, and JavaScript with frameworks like Node.js for web development. | Java, and Java-script based technologies for compatibility with Windows infrastructure. | Kotlin for native mobile development, with Flutter and React Native for cross platform apps. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: Linux is recommended for server-side hosting due to its flexibility, security, and cost effectiveness. It supports various web development frameworks and databases, making it an ideal choice for scalability. [Red Hat - Why Linux for Servers](https://www.redhat.com/en/topics/linux)
2. **Operating Systems Architectures**: The application should adopt a microservices-based architecture, allowing independent deployment and scaling of different components. This will ensure efficient load balancing, fault tolerance, and modular development. [Martin Fowler - Microservices](https://martinfowler.com/articles/microservices.html)
3. **Storage Management**: A cloud-based NoSQL database such as MongoDB Atlas is recommended for storing game data efficiently. It provides scalability, automatic backups, and global distribution for optimal performance. [MongoDB - Why NoSQL?](https://www.mongodb.com/resources/basics/databases/nosql-explained)
4. **Memory Management**: The application should utilize caching strategies such as Redis to reduce database queries and improve response times. Garbage collection and efficient resource allocation must be implemented to optimize memory usage. [Redis Labs - What is Redis?](https://redis.io/about/)
5. **Distributed Systems and Networks**: The game should leverage WebSockets for real-time communication between players and teams. A content delivery network (CDN) should be used to distribute assets efficiently. Load balancers will help handle traffic spikes and ensure smooth gameplay. [MDN Web Docs - Using WebSockets](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)
6. **Security**: The application must implement secure authentication mechanisms such as OAuth 2.0, encrypt sensitive data using TLS/SSL, and enforce role-based access control (RBAC). Regular security audits and compliance with data protection regulations should be maintained to safeguard user data. [OWASO - Authentication Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)