

Draw it or Lose it

# **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 | 06/01/24 | Jess Dowd | Overview of requirements and design constraints |

**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 is looking to develop a web-based version of their existing Android game, Draw It or Lose It. This document outlines the software design and development approach for creating a distributed, web-based game application. The new application will be designed to support multiple platforms and ensure a seamless user experience across all devices. Key features include the ability to have multiple teams and players, unique names for games and teams, and a single instance of the game in memory at any given time. The proposed design leverages object-oriented principles and software design patterns to create a scalable and maintainable system.

## Requirements 1. A game will have the ability to have one or more teams involved. 2. Each team will have multiple players assigned to it. 3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name. 4. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

**1. Distributed Environment: The application must function in a distributed web environment, requiring careful management of state and data consistency across multiple clients and servers.**

**2. Unique Identifiers: Every game, team, and player must have a unique identifier to avoid conflicts and ensure data integrity.**

**3. Singleton Pattern: Only one instance of the game service should exist in memory at any given time to manage game state effectively.**

**4. Iterator Pattern: Efficient traversal and management of collections of games, teams, and players are necessary to handle dynamic game sessions.**

**5. Concurrency: The application must handle concurrent access and updates to game data from multiple users simultaneously.**

**6. Scalability: The design must support scalability to handle an increasing number of users and game sessions without degrading performance.**

## [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) The domain model for the game application is represented by the UML class diagram. The UML class diagram consists of the following key classes: 1. Entity: A base class that includes common attributes `id` and `name`. All other classes inherit from this base class. 2. Game: Inherits from `Entity` and represents a game session. It includes a list of `Team` objects. 3. Team: Inherits from `Entity` and represents a team within a game. It includes a list of `Player` objects. 4. Player: Inherits from `Entity` and represents a player within a team. 5. GameService: A singleton class responsible for managing games, teams, and players. It uses the iterator pattern to add and retrieve games, teams, and players efficiently.

## The design leverages several key object-oriented programming principles: 1. Inheritance: The `Entity` class serves as a base class for `Game`, `Team`, and `Player`, promoting code reuse and reducing redundancy. 2. Encapsulation: Each class encapsulates its attributes and provides methods for accessing and modifying them, ensuring controlled access to the data. 3. Polymorphism: By inheriting from the `Entity` class, `Game`, `Team`, and `Player` can be treated as instances of the `Entity` type, allowing for flexible and interchangeable use. 4. Design Patterns: The singleton pattern ensures that only one instance of `GameService` exists, and the iterator pattern facilitates efficient management and traversal of collections of games, teams, and players.

**"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 can be stable and user-friendly but are less common and may be more expensive. | - Server Side: Linux servers are highly stable, secure, and cost-effective, making them ideal for hosting web-based applications. | - Server Side: Windows servers are user-friendly and widely used but can be more expensive and less secure than Linux servers. | - Server Side: Mobile devices are not typically used for server-side hosting. |
| **Client Side** | Mac clients are common and supported by many development tools. Development costs and time are moderate, and expertise is widely available. | - Client Side: Linux clients are less common. Development costs and time are generally low, and expertise is widely available in the open-source community. | - Client Side: Windows clients are very common and supported by many development tools. Development costs and time are moderate, and expertise is widely available. | - Client Side: Mobile clients require development for both iOS and Android platforms, increasing cost and time. Expertise is available but can be specialized. |
| **Development Tools** | Xcode, IntelliJ IDEA, and Eclipse can be used to develop Java applications on Mac. | - Development Tools: Eclipse, IntelliJ IDEA, and NetBeans are popular tools for Java development on Linux. | - Development Tools: Visual Studio, IntelliJ IDEA, and Eclipse are popular tools for Java development on Windows. | - Development Tools: Android Studio for Android, Xcode for iOS, and cross-platform tools like React Native and Flutter. |

## Recommendations

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

#### Operating Platform

**Recommendation:** Opt for Linux on the server side due to its renowned stability, security, and cost-effectiveness, which are critical for backend operations. For the client-side, support both Mac and Windows to cater to a broad user base, ensuring maximum reach and accessibility.

**Operating Systems Architectures**

**Details:** Linux servers are preferred for their robust security features and efficient performance, which are essential for handling high volumes of game traffic and data securely. For client systems, Mac and Windows provide diverse ecosystem support, ensuring that the game is accessible to users regardless of their preferred operating system.

#### Storage Management

**Recommendation:** Employ a relational database management system (RDBMS) such as MySQL or PostgreSQL. These systems offer proven reliability and scalability which is crucial for managing game data and user information effectively.

#### Memory Management

**Details:** Implement Java’s garbage collection mechanism to manage memory efficiently across different platforms. This approach helps in automating the memory management process, reducing the likelihood of memory leaks and ensuring that memory is used optimally in both server and client applications.

#### Distributed Systems and Networks

**Details:** Use RESTful APIs to facilitate seamless communication between the server and client applications. Design the network architecture to include redundancy and failover mechanisms, ensuring that the game remains available and responsive, even during high traffic periods or partial system failures.

#### Security

**Details:** Secure all communications between client and server using HTTPS, which encrypts data in transit, protecting it from interception or tampering. Implement robust authentication and authorization practices to safeguard user accounts and personal information. Regularly update and patch all systems to guard against known vulnerabilities.

* Follow the Open Web Application Security Project (OWASP) guidelines for secure application development.
* Employ industry-standard encryption protocols and secure coding practices recommended by cybersecurity experts.