

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.2

## 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 |
| --- | --- | --- | --- |
| 1.2 | 08/3/2025 | Jordan Belgarde | Initial draft for review and feedback |

**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 expanding its Android-based game *Draw It or Lose It* to a web-based platform that supports multi-platform compatibility. The game involves teams competing to guess images rendered from a stock library, with specific timing mechanics across four rounds.

To meet client needs, we propose a software design using **object-oriented programming (OOP)** with **Singleton** and **Iterator** design patterns. This approach ensures a centralized game management system, enforces name uniqueness, and provides a scalable foundation for web deployment. The solution is modular, efficient, and designed for future extensibility.

## Requirements

The Gaming Room requires the following features in their web-based game application:

A single game instance running in memory at any time.

Multiple teams per game and multiple players per team.

Unique names for each game, team, and player.

A web-based distributed environment to support multiple platforms.

Scalable and secure architecture for future expansion.

## [Design Constraints](#_2et92p0)

**Singleton Pattern**: Required to ensure only one instance of the GameService exists, which manages all games, teams, and players.

**Iterator Pattern**: Used to verify the uniqueness of game, team, and player names before creation.

**Web-Based Environment**: Requires stateless communication, security between layers, and platform-agnostic logic.

**Distributed Deployment**: Requires efficient resource management and synchronization between client-server interactions.

These constraints demand a robust backend capable of supporting concurrent users while maintaining state integrity and fast response times.

## [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 UML class diagram includes four key classes:

**Entity** (abstract/base class): Contains shared fields id and name.

**Game** (inherits Entity): Manages a list of Team objects.

**Team** (inherits Entity): Manages a list of Player objects.

**Player** (inherits Entity): Basic player information.

The **GameService** class is implemented as a Singleton and handles all object creation and uniqueness validation using iteration.

### OOP Principles:

**Encapsulation**: Each class manages its own data.

**Inheritance**: Entity provides a reusable base for other objects.

**Polymorphism**: Could allow for specialized game/team/player behaviors in the future.

**Abstraction**: Simplifies management of the entire game structure via GameService.

**"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** | macOS is UNIX-based with a hybrid kernel (XNU). While technically capable of running Apache or Nginx, Apple has deprecated macOS Server, making it impractical for large-scale hosting. Licensing requires Apple hardware, which limits cost-effectiveness. Virtualization is supported (Parallels, VMware Fusion), but container support is weaker than Linux. Security features include Unix permissions and sandboxing, but enterprise LDAP/Active Directory integration is less common than Windows. Cloud providers rarely support macOS servers natively, so hosting would be limited to development/testing rather than production. | Open-source, monolithic kernel (with modular support). The most common server OS worldwide, with first-class support for Docker/Kubernetes virtualization and containerization. Licensing is free, though enterprise distributions (RHEL, SUSE) can carry support fees. Security includes SELinux, AppArmor, fine-grained access controls, and LDAP integration. Fully supported across AWS, Azure, and GCP. Easiest to implement and scale as a server environment. | Proprietary, hybrid kernel. Supports virtualization (Hyper-V) and containers (Windows Containers, though Docker is stronger on Linux). Licensing can be costly for Windows Server editions. Security features include NTFS permissions, Group Policy, and strong LDAP/Active Directory support. Cloud-native on Azure, with solid support in AWS/GCP. Setup is straightforward for enterprises already invested in Microsoft stacks. | iOS and Android are UNIX-based (Darwin/iOS, Linux/Android) but **not designed for server hosting**. Their role is strictly client-side, connecting via responsive web apps or native wrappers. |
| **Client Side** | - Supports Safari, Chrome, Firefox.  - Fully supports responsive HTML5 designs. | - Fully compatible with modern web browsers (Chrome, Firefox).  - HTML5 responsive design works well. | - Compatible with Chrome, Edge, Firefox.  - HTML5 web apps run effectively. | - App must use responsive HTML5 design for mobile browsers (Chrome, Safari).  - Screen size and network variations must be tested. |
| **Development Tools** | - Required for iOS app development.  - Tools: Xcode (free but Mac-only).  - May need dedicated Mac hardware for iOS builds. | - Languages: JavaScript, Python, Java.  - Tools: VS Code, Eclipse (free).  - No licensing costs.  - Supports single full-stack development team. | - Tools: Visual Studio (free and paid versions).  - Languages: C#, JavaScript, Python.  - Higher licensing costs for enterprise editions. | - Native: Android Studio (Java/Kotlin), Xcode (Swift).  - Cross-platform: React Native, Flutter.  - Potential need for two dev teams unless using cross-platform tools. |

## 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 backend hosting due to its performance, scalability, and compatibility with Java-based servers.
2. **Operating Systems Architectures**: A 64-bit Linux architecture with Java and Docker is suitable. Containerization can isolate game instances and streamline deployment.
3. **Storage Management**: Use a relational database (e.g., PostgreSQL or MySQL) to track game state, teams, players, and history. Consider a cloud-hosted option like AWS RDS.
4. **Memory Management**: Java’s garbage collection automatically reclaims unused memory. The Singleton ensures only one game controller exists, minimizing memory usage.
5. **Distributed Systems and Networks**: RESTful APIs will manage communication between clients and the backend. Distributed architecture ensures that clients on web and mobile platforms stay in sync. Robust logging, timeouts, retries, and failover mechanisms must be built in to handle outages and latency.
6. **Security**: Use HTTPS for encrypted communication. Implement user authentication and access control. Sanitize user inputs to prevent injection attacks. Apply secure coding practices and periodic vulnerability scans. Use secure token storage for mobile platforms.