

Draw it of Lose It- The Gaming Room

# **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**

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

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

[**Design Constraints 4**](#_Toc115077322)

[**System Architecture View 4**](#_Toc115077323)

[**Domain Model 4**](#_Toc115077324)

[**Evaluation 6**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/15/2025 | Isaias Alejandro | Implemented Singleton and Iterator design patterns. Introduced Entity inheritance model. Evaluated multi-platform deployment strategies. |

**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 modernize its Android-exclusive game, Draw It or Lose It, by developing a web-based version capable of running across multiple platforms. As consultants at Creative Technology Solutions (CTS), our objective is to provide a scalable, efficient, and secure solution that adheres to best practices in software design. Our design incorporates software design patterns, including the Singleton and Iterator patterns, to ensure system integrity and proper resource management. The web-based version will support multiple teams and players, enforce unique naming conventions, and maintain a single instance of the game in memory. This software design document outlines the constraints, domain model, and technical recommendations to guide the project's development.

## Requirements

* The application must support one or more teams per game.
* Each team must consist of multiple players.
* Names of games and teams must be unique.
* Only one instance of a game can exist in memory.
* The design must support a distributed web-based environment.
* The system should ensure data integrity and minimize memory usage.

## [Design Constraints](#_2et92p0)

One of the main design constraints is that the game will be deployed in a web-based, distributed environment. This means we need to keep memory usage efficient, make sure data stays in sync across platforms, and have centralized control over key parts of the system. To handle these needs, we’re using the singleton pattern for the GameService so there’s only one instance managing game data. We’re also using the iterator pattern to check for unique names in lists. These choices help prevent data issues and make it easier to scale the game without slowing things down. On top of that, the system has to support multiple users at the same time, so thread safety and data integrity are important factors that will guide our database.

## [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 diagram shows a domain model built using object-oriented principles:

* Entity: A base class that contains shared attributes id and name for reuse across classes (Game, Team, and Player).
* GameService: A singleton class responsible for managing active games and unique IDs. It uses lists to store game instances and provides methods to retrieve or create games by name or ID.
* Game: Inherits from Entity and maintains a list of Team objects. Ensures each team name is unique.
* Team: Inherits from Entity and maintains a list of Player objects. Ensures each player name is unique.
* Player: Inherits from Entity and represents individual users.
* ProgramDriver and SingletonTester: Used to test and demonstrate the singleton behavior of the GameService.

The use of inheritance, encapsulation, and design patterns contributes to a clean, reusable, and extendable design. These principles ensure that the system is modular, supports maintainability, and meets the functional requirements 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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS is a reliable Unix-based system with solid development tool support, but it’s not commonly used in large-scale production server setups. It can be useful for local development or lightweight staging environments, but most companies opt for Linux for deployment. | Linux is a top choice for hosting servers thanks to its reliability, flexibility, and strong community support. It's also free to use, which helps reduce costs when scaling the application for thousands of users. | Windows is easy to use and widely adopted, but it can come with higher resource needs and potential licensing fees. It may also require additional configuration to support certain server frameworks and web hosting features. | Mobile platforms aren’t built for server hosting—they have limited resources and platform-specific restrictions. Hosting services are typically handled elsewhere, while mobile devices are meant to consume content and interact with the server. |
| **Client Side** | macOS needs extra testing because Safari handles things a bit differently; development can be more costly. Developers also need to ensure that cross-browser compatibility includes macOS quirks, especially for animations and input handling. | Linux offers a consistent browser experience, and being open source helps keep development costs down. It’s popular among developers who value customizability and efficient performance, especially in open-source environments. | Development is straightforward with great IDE support, but Windows-specific testing might be needed. Since it’s the most widely used desktop OS, it's important to make sure the app performs smoothly on different versions of Windows. | Essential for app development; a wide range of devices and systems means the design must be responsive and flexible. Supporting both Android and iOS requires special attention to screen sizes, gesture controls, and OS updates. |
| **Development Tools** | Common tools include Xcode, IntelliJ IDEA, VS Code, and Java SDK. Developers targeting macOS or iOS need to use Xcode for full integration with Apple APIs and testing. | Development is typically done using Eclipse, IntelliJ IDEA, VS Code, and Java SDK. These tools are well-supported on Linux and ideal for open-source collaboration and continuous integration pipelines. | Often uses Visual Studio, IntelliJ IDEA, Eclipse, and Java SDK. Visual Studio is especially helpful for developers familiar with Microsoft technologies or integrating with Azure cloud services. | Android Studio and Xcode are used, along with cross-platform tools like Flutter or React Native. These tools allow developers to build one codebase for multiple platforms, which helps save time and effort during development. |

## 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**  
   A Linux-based cloud server is the best option for hosting the game. It’s reliable, cost-effective, and scales well as more users join. Plus, Linux works great with Java applications and is widely supported in cloud environments, making it ideal for this type of project.
2. **Operating System Architecture**  
   Most Linux servers run on x86\_64 architecture, which is compatible with modern web frameworks and optimized for Java. It supports both vertical and horizontal scaling, so whether you need to beef up a single server or add more servers to handle growth, this setup can handle it.
3. **Storage Management**  
   To manage the game’s image library and user data, a cloud storage service like Google Cloud Storage or Amazon S3 is recommended. These platforms offer secure, scalable storage that can grow with the app. They also come with helpful features like automatic backups and fast access from any device, which makes storing large files much easier.
4. **Memory Management**  
   The Java Virtual Machine handles memory pretty efficiently by managing allocation and cleaning up unused data through garbage collection. To keep memory use low, the app can also use the Singleton pattern, which ensures only one instance of the game is loaded at a time. This helps keep things running smoothly, especially when images are being rendered quickly during gameplay.
5. **Distributed Systems and Networks**  
   Breaking the app into microservices is a smart move—it separates different parts of the system like user management, game logic, and image handling. These pieces can talk to each other using RESTful APIs. Hosting the services on the cloud adds reliability with tools for load balancing, failover, and system monitoring, helping the app stay online and responsive even if one part runs into trouble.
6. **Security**  
   To keep user data safe, all communication will go through HTTPS so it’s encrypted. We'll set up proper authentication and authorization to make sure only the right users have access to the right parts of the app. Data will also be encrypted when stored and while being sent. On top of that, adding firewalls, intrusion detection, and activity logs will help keep the system secure and give visibility into any suspicious activity.