

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 | 09/21/2025 | Ryan Jones | Updated the requirements and 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 expanding Draw It or Lose It into a web-based application. The solution is a Java backend application service that enforces one in-memory game service. The backend must guarantee unique names for players, teams and games. Each game supports one or more teams; each team consists of multiple platers.

## Requirements

* *Support multiple teams and players per team*
* *Enforce uniqueness*
* *Maintain one instance of the game service*
* *Assign unique IDs to each game, team and player*

## [Design Constraints](#_2et92p0)

* Network latency and potential connectivity issues requires validation to take place server-side
* Single service to maintain state of the games
* Should perform well on all web browsers

## [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)

* Entity base class, id, name. getters and setters
* Game Service: hosts a list of games and the IDs for the next games, teams and player. Provides getInstance(), addGame(), getGame()
* Game creates and stores each unique team
* Team creates and stores each unique player

**"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** | Technically capable but macOS is not commonly used in production due to its hardware and licensing constraints | Preferred for production. Performant, large ecosystem. Excellent JVM support | Viable with good support for JVM and has IIS. Licensing costs make using Windows cost prohibitive | Doesn’t work well as a server due to limit hardware capabilities |
| **Client Side** | Safari requires extra QA | Web clients fully supported but with a very small user base | Largest desktop user base. Web clients work well | Making a responsive web app can make these viable. Great native support |
| **Development Tools** | JDK 17+/21, IntelliJ IDEA, Gradle, Docker Desktop | JDK 17+/21, IntelliJ IDEA, Gradle, Docker Desktop | JDK 17+/21, IntelliJ IDEA, Gradle, Docker Desktop or WSL | Standard web toolkits |

## 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, offers the best cost and stability and is widely supported by cloud providers
2. **Operating Systems Architectures**: Stateless application wrapped in a docker container for ease of use and future scaling via tools like Kubernetes.
3. **Storage Management**: use Postgres for state management. Though the current design suggests in-memory state this would become an issue if the game instance crashes
4. **Memory Management**: optimize the JVM Garbage Collector to keep the object count under control in order to keep overall memory utilization tightly confined to only what the application requires
5. **Distributed Systems and Networks**: Expose the service over HTTPS, using JSON APIs. Plan for network inconsistency in the player base by adding retry logic
6. **Security**: Enforce server-side validation of all creation and naming. Sanitize inputs