

**Draw It or Lose It**

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 10/19/25 | Joshua Kelly | Finalized for Project Three submission |

**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)

Draw It or Lose It was asked to be created as a multi-platform, web application by Gaming Room. It needs to place unique identifiers for games, teams, and players and keep active game data in a single service. My solution applies a singleton design pattern so there would only be one instance of the game service, and inheritance and iterator use so entities are encapsulated neatly. It keeps the code efficient and flexible and enforces the business rule of uniqueness throughout the whole system.

## Requirements

Draw It or Lose It business requirements include scaling Draw It or Lose It from a single platform to a distributed web application supporting numerous clients, gaining broader reach, and achieving cost-effective scalability. Technical requirements include supporting multi-OS, enforcing separate game, team, and player names, and centralizing data management by a single-service solution. It is further required of the application to allow for smooth extensibility for future growth and retain uniform performance across client platforms.

## [Design Constraints](#_2et92p0)

As a distributed and web application, it must manage concurrency and stateless requests. Its singleton usage ensures there is a sole instance of the services that manage games, but this imposes limitations around thread safety and scalability for growing numbers of users. Another limit is platform independence; the application will have to run on different operating systems and devices, so development will need to employ portable languages and frameworks. Finally, there needs to be uniqueness of entity names throughout the entire system, which imposes limitations around the way we structure lookups and requires extra validation logic utilizing iterators.

## [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 exhibits a base class, Entity, that contains an id and a name. Game, Team, and Player extend Entity, exhibiting inheritance and code reuse. GameService is a singleton and contains a list of games. A game can contain many teams, and a team can contain many players, exhibiting aggregation. Iterators are used in lookups so entity lists may be searched at constant time while maintaining encapsulation. These design principles of inheritance, encapsulation, and design patterns work together to enforce the business rule of uniqueness of identifiers and provide a clear, maintainable structure.

**"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** | Rock-solid and developer-friendly, but production hosting is not typically done on a Mac. It does have a built-in web server, however, and it's expensive and not very scalable compared to a Linux solution. | It is the most widely used solution for hosting because it is free, open-source, secure, and extremely scalable. It is highly community-backed and widely used on servers, too. | Server versions of Windows operate smoothly in business environments but require licensing fees. They're dependable but may consume more resources than Linux. | Server hosting is not appropriate for mobile devices. Mobile devices lack the stability, security, and always-on needs of production systems. |
| **Client Side** | Developing for Mac-based clients involves cross-platform interoperability assurance, but after creation, maintenance cost is acceptable. There's a need for developer experience in Mac-specific testing. | Linux client support is uncommon but valuable for power users. It's cheap, but it might demand more effort in terms of testing since there's a large number of different Linux distributions. | It's the most used desktop platform, so comprehensive support is most important. Most development and testing effort goes here since it's the most used platform. | It takes more development time to support Android and iOS, but you need it if you want broad reachability. It incurs expense but significantly increases accessibility. |
| **Development Tools** | Java, Eclipse IDE, and IntelliJ IDEA work smoothly on Mac. | Same toolset as Mac; Linux is ideal for open-source development. | Visual Studio Code and Eclipse do work; Java is typically supported on Windows. | Android Studio for Android and Xcode for iOS, used in tandem with cross-platform solutions if needed. |

## 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**: Server-side hosting is most effectively carried out on the Linux operating platform since it's cost-free, secure, and extremely supported.
2. **Operating Systems Architectures**: We recommend adopting a multi-tier architecture for the platform, having application and database running under Linux servers and clients on Mac, Windows, and mobile platforms.
3. **Storage Management**: A relational database such as MySQL or PostgreSQL would be suitable for enforcing unique identifiers and persistent storage management.
4. **Memory Management**: Java Virtual Machine (JVM) manages memory automatically through garbage collection, and memory consumption is efficient for Draw It or Lose It.
5. **Distributed Systems and Networks**: Standard internet protocols (HTTP/HTTPS) will be utilized in the application. Redundant servers and load balancers will be used to reduce the possibility of failure. It will allow different client platforms to communicate with the same service.
6. **Security**: Security will be managed by encrypted connections (SSL/TLS), authentication processes, and proper data processing. Sensitive information will be stored safely, and accessibility control will be carried out evenly across platforms.