

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 | 09/27/24 | Nash Ellis | First Draft |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to develop a web-based version of their mobile Android game called “Draw It or Lose It”. The game is an optional multiplayer / multiteam game where players will guess the puzzle (phrase, thing, or title) using rendered stock images stored in a large library within the game. The game lasts 4 rounds, with each round lasting a minute.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *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)

* Singlton Pattern is necessary to ensure one instance of GameService exists at a time.
* The iterator pattern must be used to get instances and iterate through instances to avoid duplicate names/gamertags or teams.
* Since the game will be web based, it should be accessible to all Computer-based OS.
* Database will need to have unique identifiers for players, games, teams, and perhaps images within the library.
* The library of images must be very large and continue growing to not be boring. After many games played, repeated images would ruin competition and enjoyment.

## [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 is used to represent the domain model. The UML includes classes for Game, Team, Player, Entity, ProgramDriver, SingletonTester, and GameService. The Entity class has common attributes from the Team, Player, and Game class through inheritance. The ProgramDriver class has an association with the SingletonTester. The GameService has a zero to many relationship with the Game class, the Game class, has a zero to many relationship with the Team class, and the Team class has a zero to many relationship with the Player class.

**"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 is known for its security, stability, and reliability. It isn’t commonly used for server-side things like web hosting, therefore has a lack of support and is expensive. | Linux is practical for servers, offering open-source flexibility, stability, and scalability for hosting. | Windows is best utilized for Microsoft systems/environments. Can be more expensive than Linux. | Mobile Devices can host servers using Linux on Android and can be very compatible. |
| **Client Side** | Mac development requires extra effort and is more expensive but ensures a smooth user experience for Mac users. | Linux client use isn’t very common but is useful for technical clients and development. | Windows clients are most common. Windows offers compatibility and low development costs. | Mobile app development for iOS and Android is very important, since mobile users dominate the market. |
| **Development Tools** | Mac development uses Xcode for native apps, with other tools like IntelliJ for web apps. | Linux supports a variety of languages and IDEs like Eclipse and IntelliJ, making it highly flexible. | Windows development uses Visual Studio for .NET apps and supports languages like C# and Java. | Mobile development involves tools like Xcode for iOS and Android Studio, or cross-platform tools like React Native. |

## 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**: I would suggest Linux as the best option for The Game Room to create their game. Linux is a cost-effective (cheap) option with great flexibility, performance, and scalability.
2. **Operating Systems Architectures**: Linux is very compatible with MySQL and Python, while also having other programs it can benefit from. MySQL will be ideal for storing the vast amount of data needed, and it can be scaled.
3. **Storage Management**: As previously mentioned, since the storage required is large, I suggest MySQL. SQL is easy to use and navigate, while being capable of storing all the data you need and growing with your needs. An alternative would be AWS with its cloud storage.
4. **Memory Management**: Linux is great at handling memory with dynamic paging. Linux is known for its stability and smooth performance.
5. **Distributed Systems and Networks**: As mentioned previously, AWS is an option if the company sees it fit. REST APIs would be ideal for The Gaming Room's game, as they allow efficient communication between clients and the servers. Creates “Seamless Interaction” between platforms.
6. **Security**: Linux’s built-in security features are already great, including iptables and SELinux. These can also be paired with SSL/TLS encryption and role-based access control to ensure a secure environment for user data.