

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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 client, The Gaming Room, would like to create a web-based version of their current Android game: Draw It or Lose It. Draw It or Lose It is a multiplayer picture guessing game. The web-based game needs to several platforms not just Android. The Gaming Room has outlined some of their requirements for the game which are described in more detail below in the design constraints.

## [Design Constraints](#_2et92p0)

* Draw It or Lose It must be a web-based application and run in multiple different environments (Mac, Windows, Android, iOS, etc.)
* Draw It or Lose It must utilize network connections to allow for multiple users to connect to the same game (hence the multiplayer aspect)
* Draw It or Lose It must have unique team names and must allow the users to check the validity of their name before submitting it for themselves
* Only one instance of the game can exist in the system memory at any given time so all players will be connected to the same game.

## [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 Entity Class below serves as a base class for all objects in the game. Entity contains the property’s ID and name, which each object will use to identify itself. The game, team, and player classes will all extend from the Entity class so that they may inherit its properties. The GameService Class will serve as the system manager and will coordinate and drive individual game instances. The GameService may be related 0 to many instances if Game objects. The game class is related to the team class in the same manner and may be related 0 to many instances of Team objects. The team class will also carry this same relationship to the player class. Lastly the ProgramDriver class initializes the one and only instance of the GameService, as well as using the SingletonTester class to perform validation at run-time.

**"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 can be used as a server however licensing can get expensive. Additionally, MacBooks will be required to develop | Linux is very well-equipped for hosting a web based application and is the one of the most popular. Additionally, licensing is Free | Windows has a server which is very easy to set up and is very secure. However, licensing can get very expensive | Even though Mobile devices Can be used as Servers they are Poorly equipped to support high-end applications. However they can be used for Development. |
| **Client Side** | Mac has good and easy to use SDKs but again you must have a MacBook to develop for Mac. | There would be an increase in development cost here being that it would take more time and you also need someone who is experiences at Python. | Expertise is the biggest requirement for Windows. In this case the .NET framework is the best choice both for its security and capability. | In this situation, it is best to find developers who have experience in developing apps. This is important because how things are displayed need to be done differently than on the web. |
| **Development Tools** | A MacBook with iCode on it would be a great tool in this instance. All coding would have to be done using Swift. | Python comes already installed on most Linux machines. One of the more suggested IDEs for coding Python is IntelliJ’s Ultimate IDE but personally I used Eclipse and Visual Studio | Visual Studio is the standard and best way to code in windows. It provides a seamless way to code in any language including C++ or C# which most Windows Programs are wrote in. | There’s three options to go with in this situation. You either need someone who specializes in android studio to develop the app, someone who can code the app for iPhones in iCode, or someone who specializes using Unity which is coded in C++ then converted to android and iOS. However, keep in mind that a Mac is required for the iOS Conversion |

## 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 using a Linux Ubuntu sever to host the game if a Kubernetes cloud setup. Linux offers the easiest way to code and preforms really well as a hosting service. They allow you to control who plays your game. For example, if you encounter a person who is found to be cheating at the game and you want to ban them from your games, hosting through Linux would make this a lot easier to remove him and ban him from the server. Overall as a gamer myself, a lot of people who host dedicated servers or custom servers would agree that Linux Ubuntu is by far the best for hosting game servers
2. **Operating Systems Architectures**: Linux kernel is very secure and stable and the Kubernetes clusters will allow the system and hardware needs to be efficiently separated. Linux has everything that you will ever need in getting the server up and running. Linux is known for its low latency which makes it a good choice for game server hosting. It is important for the application to be both manageable and easily changeable which I think the Cloud setup will allow for these things to be done easier.
3. **Storage Management**: I feel that a SSD being used for storage here would allow for faster access to items allowing for a more seamless experience for the user. I suggest using a Kubernetes node for storage of files and an NoSQL node for user management and data management to optimize storage. Especially in a game where photos are being loaded it is very important for fast access to storage which a SSD will make files more easily accessed in comparison to a HDD. The photos in this case need to be able to be accessed as fast as possible or it could slow the game or cause issues.
4. **Memory Management**: To be more cost effective, I feel that the addition of a watcher for the load on the system. This would mean that when usage is low the required memory can be lowered. Meaning that it can conform the required memory to the workload meaning in simple terms that you are “Only paying for what you need”. The Kubernetes cloud aspect of my suggestion would allow for this to be done rather easy. Optimizing both workload and costs attributed to the number of users within the game. This would save the server owner money while also providing the best possible experience for the user.
5. **Distributed Systems and Networks**: Being that the system will be on the cloud. Required maintenance on servers would be able to be performed without shutting down the game. This also allows another server to start automatically if the system crashes. Overall this provides an easier management and organization of the system. Personally the most annoying thing as a gamer is coming home after a long day and loading up your favorite game just to be met by a screen that says that the game will be down for the next several hours for maintenance. The application being on the cloud will take the need for shutting down out and provide users with a great experience as well.
6. **Security**: I suggest that a role-based security system be used. It will allow for the separation of the roles of Admin, Game, Team, Player, And User. With this you control access of certain information by certain roles. This prevents the possibility of “Modders” changing the game to gain an advantage. Also this protects the game and its users in the fact that if a user has won 321 games. Security prevents someone from maliciously accessing the server and deleting the user. In a broad sense the addition of security protects both the user, the game, and the server.