

Draw It or Lose It Game

# **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 | 06/16/2023 | Brennan Riley | Complete software design documentation for Gaming Room multi-platform web application |

## [Executive Summary](#_sbfa50wo7nsh)

The client company The Gaming Room has requested a web hosted platform for their Draw It or Lose It game. The criteria given by the client state that the application must support multiple players on multiple teams as well as storing customized names for these players and teams. The software then checks new names against existing names. Only one instance of the application should be stored in memory at a given time.

## Requirements

* Web-based software written in Java for ability to replicate across multiple platforms
* Game should check player and team names against previously stored names
* Game should generate unique game instances and store only a single instance in memory at a time
* Game should associate custom ID for players, teams, and game instances
* Game should store player and team variables as object types

## [Design Constraints](#_2et92p0)

* Game will need to be accessible across multiple platforms and multiple devices
* Game needs to be able to store increasing amounts of data over its lifetime
* Game will require commitment of resources for initial development as well as fur supporting the application long-term

## [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 diagram below describes the architecture and the relationship between classes within the Draw It or Lose It game. The Entity class here serves as the parent to the Game, Team, and Player classes. These three classes therefore inherit the features of the Entity superclass. Anything required by the subclasses can be written a single time in Entity. The “0 to many” association between the GameService, Game, Player, and Team classes allows these classes to utilize multiple instances of each other depending on the end user input. ProgramDriver and SingletonTester are independent classes which work to drive the main Entity and conduct ID verification on user inputs.

"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** | Apple utilizes Mac OS X Server for around $30 a month per server. It is relatively low cost and scalable but is not particularly suited to multi-platform hosting. | Linux possess multiple different multi-deployment options such as RedHat. It is reasonably priced, and its format allows specialization. Linux however is relatively unpopular amongst end-users | Windows Server systems are an extremely popular option. They are one of the more expensive options, but this expense includes a number of tools and support options. It is widely used among end users as well. | It is likely not possible to host the server on a mobile platform. There are currently no mobile devices with the computing power required to support a multi-platform gaming server. |
| **Client Side** | Due to the proprietary nature of MAC OS X and its systems. Cost has increased due to this and therefore cost should be considered in resource allocation. Users are also familiar with the platform and it has a shallow learning curve. | Linux server systems would probably be the least costly option. Linux fluent development teams will be necessary as well as needing a customized Linux solution. | Windows would likely be the most expensive option but also the easiest to deploy and support. Development would be efficient due to the wide tool kit and development suite. A custom systems package will also need to be designed. | Due to the decreased support capability, mobile platform work should be focused on user experience. These systems will likely not be able to support server funtion needs. |
| **Development Tools** | Swift is the primary language of any development for the Mac OS platform. Swift can also be developed and deployed from many different environments. | Development environments such as Eclipse can be used to code for the Linux platform. Languages such as Java, JS, and the C family can be utilized. | Visual Studio is a very common Windows software development platform. Visual study supports multiple languages as well as project formatting solutions. | The most common mobile platforms are the iPhone and the Samsung. iPhone applications utilize the swift language while Samsungs typically use Android Kotlin. |

## Recommendations

1. **Operating Platform**: The best candidate for an operating platform in this case is the Windows OS. This is due in large part to its ability to be deployed to and supported on multiple different platforms. They also provide a wide suite of tools and support mechanisms for the deployment and maintenance of the Game Room software application.
2. **Operating Systems Architectures**: The primary layer of the Windows OS is the Windows API which allows users to interface with and manage the rest of the system. Using this primary layer, the GUI is generated for the application running on the system. The OS follows a more traditional monolithic pattern with information passed up and down the system with the assistance of its internal tool set. The API also assists in the allocation of additional resources and provisioning of needed storage space.
3. **Storage Management**: Along with the Microsoft OS, I would recommend the utilization of the Microsoft Azure Cloud. This system is a highly effective way to create an efficient and adaptable storage system. In addition to basic storage, there are a number of automation and update solutions included on the platform. The storage can also be quickly scaled and altered depending on the needs of the organization and the application.
4. **Memory Management**: The memory system for this game will need to be able to handle a relatively large amount of initial data. Along with the 200 original images, it will need to be able to handle an expanding archive of player and game data. This will include passwords, usernames, and team names. This information will be backed up and maintained at the storage level on the host server storage.
5. **Distributed Systems and Networks**: In the case of this application, it will likely be best to use a parent-child system. This is to say that a main host server, which is monitored by the organization, will share access and generate instances of the game upon receiving a request from the end user. This instance contains surface level access such as game information and the UI while the main operations of the game are still being handled within the main parent server.
6. **Security**: Although the Windows OS includes several built-in security features, there are several steps that can and should be taken to improve the system. A dedicated network security department is a great internal practice that provides much better oversight and an ability to actively prevent breaches. The integration of a multi-layer permissions and access management system is another good way to prevent people accessing systems that they do not require access to. In addition, firewalls at multiple levels further prevent unauthorized access even in the case of a breach within the organization.