

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 | 11/19/2024 | Nicholas Gard | Added initial executive summary, design constraints and domain model. |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

Our client The Gaming Room is looking to set up a web-based game based on the television game Win, Lose or Draw. The application will render images for players to guess from a large library of stock drawings. The game will involve one or more teams consisting of multiple players. Only one instance of the game should exist at a time so each team and player should have a unique identifier.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

* 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.
* Drawings are rendered at a steady rate and are fully complete at the 30-second mark. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit.
* Web-based

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

The Gaming room already has an Android only application that they wish to develop anew as a web-based one. The new application will be developed in Java for its web capabilities and will connect to a library containing all possible drawings for the game. It will need to have one instance accessible from multiple devices to incorporate teams and players into a game. Java is the primary language for Android development so it should be simple to transition to the web.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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.**

The application consists of a main driver class that will be used to initiate the creation of the games, teams, and players. The actual making of the game is through the GameService class and follows a singleton design pattern so that only a single GameService class may exist at any time in memory.

GameService blocks its creation of any instances by setting its constructor to private. The only way to instantiate a GameService is through the getInstance() method. getInstance() checks whether GameService has been started and only begins if it is not present in memory.

Once GameService is running, the driver class is able to call the method addGame(). addGame() uses the iterator pattern to prevent similarly named Game objects from being created. This new Game object is then added to the list games.

After the game is created a team may be added to the it with the addTeam() method. addTeam() uses the iterator pattern to prevent similarly named Team object from being added to the game. This new Team object is then added on to the List teams.

After the team is created a player may be added to the team with the addPlayer() method. addPlayer() uses the iterator pattern to prevent similarly named Player objects from being added to the team. This new Player object is then added to the List players.

Game, Team, and Player classes are all subclasses of Entity. Entity has 2 protected attributes: id and name. The default constructor is also protected so null objects are blocked at creation and only the overloaded constructors may be used.

The UML diagram demonstrates multiple OOP techniques. Polymorphism and inheritance are used when we extend the Entity class into multiple sub-classes and overload the constructors. Encapsulation is used multiple times in classes with both methods and variables. Abstraction is used in the GameService class and others to hide data with private attributes.

## [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** | The OS X server is available for Mac computers. Finding hosts is difficult and expensive. A typical server is $499 or $999 for unlimited data. | Linux is the most popular operating system for webhosting. Typical cloud hosting is done through Linux. It is also open source so it can be very cheap. | Windows is often the easiest to set up since it is so commonly used. Licensing costs are typically high since it is the most popular option. A server can range from $500 to $6,200 based on the size. Compatibility is much better with windows as well since it is the most popular OS. | The hardware is very limited for mobile devices, so they are not great for hosting. They work well for individual use but are not equipped for multi-user hosting.  Costs would exceed all other options and would have to be custom built. |
| **Client Side** | Mac uses lesser-known languages for programming. The MacOS SDK is in Objective-C or SWIFT. 75% of the market uses Windows while only 16% use Mac. | Linux development is more straightforward where you can use java or C/C++ or Python. Multi-user support if available on GNU/Linux | Windows has been a native multi-user platform since Windows XP. Windows is typically developed using C# and the .NET framework. There is not much of a barrier to entry as C# is a common language. | The Android SDK is based in Java so the is no barrier for coding Android devices. iOS is programmed in SWIFT so could pose more challenging but also represents the bulk of the market. Mobile devices aren’t good for multiple users but developing a client application should be simple. |
| **Development Tools** | Mac uses Objective C and SWIFT for coding. Xcode is the most used IDE. Xcode costs $99 for the basic developer kit. | Linux uses C/C++, Python and Java for development.  Each language has its own common IDE’s Python- Pycharm, Java- Eclipse, C++- Visual Studio.  All of these IDEs can be used for free. | Windows is developed using C# and the .NET framework.  Microsoft Visual Studio works very well with .NET and C#. It is very feature rich and easy sets up .NET apps.  Visual Studio and .NET are free to use on Windows. | Android SDK is based in Java and most apps are developed in Android Studio. Android studio is a free IDE.  iOS uses mostly SWIFT for Mobile apps but can also use Objective-C. Most are written in Xcode which is $99 per year. |

## 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**: For the back-end systems the best Operating Platform is clearly Linux. The low cost is a huge perk and doesn’t limit access to data centers. Even though not many users have Linux we will be connecting to clients via API anyway. Linux has great, customizable security options with tools such as Selinux, ruptime and many open-source options. Since the frontend is agnostic, we can program in the preferred language for the platform.
2. **Operating Systems Architectures**: For architecture I would suggest a backend that manages the game environment, and the frontend only handles rendering. Since Draw it or Lose it does not rely on fast load times and transmission can be done asynchronously with little caching. A cloud server works best for scalability and could run containerized microservices. Kubernetes would work well as an orchestration system allowing for easy, scalable deployments.

Using the front end for rendering lets the server offload some of the more resource heavy parts of the application which helps reduce the hosting costs. The client can also cache data from the server. This lets it maintain a consistent frame rate and timing. This might be easiest to implement with a browser-based application to let users easily access it.

1. **Storage Management**: Since we plan to use Cloud based storage no discussion around physical storage is needed. If we plan on building our own server, we need to decide between SSD and HDD. HDD is good for long term storage and SSDs would help with fast read/write speeds.
2. **Memory Management**: Linux makes use of pagecache for data in the main memory or virtual memory for any pages allocated. It also uses a technique called demand paging to ensure that pages not actively in use are not loaded in memory. This is based on the Least Recently Used algorithm.

Android Runtime and Dalvik virtual machine use paging and memory mapping to manage memory. This makes it so that any memory an app modifies remains resident in RAM and can’t be paged out.

1. **Distributed Systems and Networks**: Applications are being built on cloud native architectures more often because they offer better uptime and have almost no outages. Most cloud providers can replicate and shift services among different deployments to prevent outages.

We will use a RESTful API to communicate asynchronously from backend to frontend. This

allows transparent and secure communication from the clients and the server regardless of OS.

1. **Security**: Protecting user information is a high priority so we will be using role-based authentication to sequester access to important systems. Because of this we will need to create an entitlement interface to assign roles to users. Implementing the idea of least privilege will help keep the system secure. Most users will only have access to game systems and roles. While a handful of administrators will have full access to user records and controls. The API will be protected using encryption as well in case traffic is intercepted. Linux has a suite of tools we can use to set up a firewall and protect user data.