

Draw It or Lose It

# **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 | 5/25/2022 | Lance Heatwole | New client application. No previous versions. |
| 2.0 | 6/4/2022 | Lance Heatwole | Added Evaluation Section |
| 3.0 | 6/25/2022 | Lance Heatwole | Added Recommendations Section |

**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 Gaming Room currently has an Android version of their game Draw It or Lose It. They are wanting to expand the game to a web-based version for multiple platforms. The Gaming Room does not know how to create the environment for the game.

## [Design Constraints](#_2et92p0)

* The game must be able to function on any device using any platform.
* The game must only accept unique identifiers for game, team, and players.
* Must allow for multiple users in each game to be on different platforms.

## [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 ProgramDriver class contains the main method for the program and uses the SingletonTester class to test for a single instance of game, team, or player. The entity class is the parent class to the game, team, and player classes and inherits attributes from them. The game, team, and player classes contain a single unique game, team, or player at any time. The GameService class creates a unique game, made of unique teams, made of unique players.

**"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** | Macs can be used as servers, however, there is the pricing issue of licenses which can be expensive. | Linux is a free OS and has free licenses making it the most popular option. | Windows is also useful as a server. It is ubiquitous and familiar to most users. It is also expensive for licensing. | Mobile devices are not recommended as servers. They can be used, but lack the processing power that may be needed for a web based application. |
| **Client Side** | Development for Macs would require Macbooks to write code on. Each Macbook will be a sizable cost requirement. | Linux may be the most demanding OS to develop for. However, any PC is capable of running Linux. All of the source code is freely available. | Windows may be the easiest to find developers for. It’s the most ubiquitous of the operating systems mentioned. Any PC can run Windows. | Mobile Devices are generally different than developing for a PC. They have size constraints and processing constraints that need to be accounted for. User interaction will also vary by device. |
| **Development Tools** | Programming language would be Swift since its Mac using Xcode as the IDE. | For Linux you would be developing in Python which should be included. Would recommend PyCharm. Theres a free and premium version. | Windows would require Visual Studio most likely since you’d be coding in C# or C++ which are the standard. | Android Studio for Android devices and a Macbook for Swift and Xcode for iOS. |

## 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 recommend a Linux Ubuntu operating system with a Kubernetes cloud setup
2. **Operating Systems Architectures**: Linux kernel is stable and secure. Kubernetes allows for efficient separation of hardware and system needs.
3. **Storage Management**: Either hard disk drives or solid state drives are appropriate. The preferable choice would be solid state drives. They allow for faster access to elements needed for the application. The price for SSDs will be a bit higher, but the tradeoff for faster application speed is of paramount importance.
4. **Memory Management**: I would recommend setting up a watcher on the memory load. That way during slow times, less memory can be allocated, and during peak times the opposite. This will decrease the cost incurred for memory usage long term.
5. **Distributed Systems and Networks**: Since the application will be running on the cloud, this will allow for servers to switch on and off easily. Therefore, if maintenance is necessary on one server, another can take over without loss of game time for the user. This also allows for servers to jump in at anytime during peak hours when other servers are overloaded.
6. **Security**: The recommendation would be for a role based security system for admins and users both. This way users will be unable to access sensitive admin areas and the application will remain unaffected. This also means that new roles can be added if necessary, and their areas can be segregated as well.