

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[CS 230 Project Software Design Template 1](#_Toc2043592311)

[Table of Contents 1](#_Toc1860225656)

[Document Revision History 2](#_Toc681365267)

[Executive Summary 2](#_Toc2107439198)

[Requirements 3](#_Toc1179823128)

[Design Constraints 3](#_Toc1196531729)

[System Architecture View 3](#_Toc1177395483)

[Domain Model 3](#_Toc1316006435)

[Evaluation 4](#_Toc1978876967)

[Recommendations 5](#_Toc872737948)

## 

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/31/2024 | Alex Barber | Initial creation of the software design |
| 1.1 | 04/14/2024 | Alex Barber | Populated evaluation section |
| 1.2 | 04/28/2024 | Alex Barber | Added recommendations |

**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)

A new client, The Gaming Room, wants to develop a new web-based game that will be available for multiple platforms. They would like to base it off their current game, Draw It or Lose It. Currently this game runs on Android, but The Gaming Room would like to expand to other platforms such as iOS, Windows, Linux, and MacOS. The game consists of four rounds of play lasting one minute each. The game will slowly render images from a stock photo database. One player team will have 30 seconds to solve the puzzle before the image fully renders. If the players do not guess the puzzle in time, the other player team will have 15 seconds to solve the puzzle.

## Requirements

* A game can 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.

## [Design Constraints](#_2et92p0)

* Cross-Platform: We will need to translate the game to a webapp so that players from multiple different platforms can access the game and play with friends from different platforms. All platforms have different software development kits.
* UI: We need to translate the already finished Android app in a way that is familiar and true to the design language of the original game.
* The API must be tailored to function on multiple platforms.
* Use unique IDs for each instance of a game, team, and player to limit instances of the game to one.
* Licensing for images used in the 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.

## 

The ProgramDriver class contains the main method. ProgramDriver then uses directed association with the SingletonTester class to test if any instances of GameService are already running. The Game, Team, and Player class all inherited from the Entity class. This makes the Entity class a parent class. A Player cannot have a Team, but a Team can have a Player. A Team cannot have a Game, but a Game can have a Team. A Game cannot have a GameService, but a GameService can have a Game. GameService can only have one instance of a game running at any given time. Each Game can only have one unique Team playing at any given time. In the same vein, each Team can only have one individual Player at any given 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**

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 must 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 a server, but licensing is expensive. Alongside the higher cost for hardware, it can be a costly platform for running the server. | Linux is very popular for web-based hosting because it has great tools. Licensing is free as well. | Windows is the most common platform, and as such it can be easy to get servers running, but licensing is expensive. | Mobile is not as well equipped as the other platforms for server use. They lack the power needed for a larger server operation. |
| **Client Side** | MacOS has easy to use SDK’s. However, we must buy Apple hardware to develop for Apple products. This will add cost. Experience in Swift is necessary. | Experience in Python is required. Development times may be longer on this platform. | A good understanding of the .NET framework would help minimize labor costs. | Developers with experience creating mobile apps are preferred to minimize labor costs. Time will be a little more strenuous to integrate the app to multiple display configurations needed for the app to function across multiple different devices. Mostly with different Android devices. |
| **Development Tools** | MacOS employs their own coding IDE called xCode to write applications in Swift. IntelliJ offers their suite for other languages. | Python is included in most Linux distros. You can also install IntelliJ’s suite for IDEs for Java, C++, etc. | Microsoft’s own Visual Studio can be used to code Windows Applications. IntelliJ also offers their suite for other languages. | Android Studio can be used to develop for Android. For iOS, a developer must have an Apple computer to utilize xCode and use Swift. |

## Recommendations

Analyze the characteristics of techniques specific to various systems architectures and recommend to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: I believe that running the game using a cloud-based or serverless architecture using a Linux distro would be the best approach for The Gaming Room. This will help relieve a lot of costs from the company and allow for flexibility as the game scales. This approach negates the need for The Gaming Room to purchase hardware to sustain the servers for the game, while allowing for higher allocations of hardware to be employed if needed as the player base grows.
2. **Operating Systems Architectures**: The kernel used by Linux is stable and secure. The OS is also built on a modular framework. The heart, or kernel, of the OS contains only core components while separate modules promote dynamic expansion of services (Silberschatz, 2009). This means that using Linux would allow for disk usage to be decreased as the OS would only need to have necessary modules installed which minimizes bloat on the disk. Linux also employs system libraries that do not have direct access to the kernel, which provides an extra layer of security.
3. **Storage Management**: Costs for SSD applications in servers have decreased drastically recently. SSDs allow for faster asset access, and faster load times for the images (even though this may not be as important with the extended render process). The use of HDD’s would be cheaper but may lead to a lesser user experience.
4. **Memory Management**: With a Linux based server, we can take advantage of Linux’s partial memory execution through avenues such as virtual memory. This can assign RAM allocations to separate addresses in the server. This allows the server to allocate the RAM efficiently to processes that may need more or less RAM. Each address’s RAM is inaccessible or invisible to the other processes. This will eliminate constraints from memory availability and allow for multiple programs to run in parallel. This will also allow for faster loading times.
5. **Distributed Systems and Networks**: Because this would be a cloud-based game, if there is required maintenance on the servers, the game would not have to be shut down. It can simply be moved to another node on the server. This also allows for redundancy to keep the game running in the event of a server crashing. The Gaming Room should consider developing an API to serve clients from most platforms that would be expected to access the game. REST API would be a great solution. A cloud-based approach will allow for all required elements for each respective platform to be hosted in the cloud. This allows all operating systems to have a client made to access their necessary resources.
6. **Security**: I would recommend a role-based security system. This allows for separation from admin, player, and user. This would prevent the users from accessing information they should not. Each role should have its own layers of authentication and authorization. I would recommend stricter rules for authentication in the admin level of authorization. I would recommend 2 Factor Authentication(2FA) at the minimum. The player and user role would allow for authorization to play games, manage teams, and manage their profiles and accounts. This measure should keep the players in a sandbox, while admin roles have access to managing file libraries, code, editing puzzles, and access to user accounts/profiles.

**Sources**

Silberschatz, Abraham, Peter Baer Galvin, & Greg Gagne. (2009). *Operating System Concepts, Eighth*  *Edition.* John Wiley & Sons, Inc. https://learning.oreilly.com/library/view/operating-system- concepts/9780470128725/.

The Linux Foundation. (2017, April 4). Introduction to Memory Management in Linux [Video]. YouTube. https://www.youtube.com/watch?v=7aONIVSXiJ8