

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 | 03/24/24 | Andrew Miller | Added Executive Summary, Design Constraints, Domain Model |
| 1.1 | 4/7/24 | Andrew Miller | Added Evaluation |
| 1.2 | 4/17/24 | Andrew Miller | Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room currently offers an Android-exclusive app; Draw It or Lose It. They seek to develop a multi-platform web-based game based on the current version of the game. The game should last for four rounds, each one minute in length. The game should have only one instance at a time, with each instance containing at least one team, each team with multiple players. Game and team names should be unique.

## 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.>*

## [Design Constraints](#_2et92p0)

Game is intended to serve multiple platforms, so must be compatible with (at least) Android, iOS, and Windows devices.

Each platform will have its own development kit, and may require specialization in each kit to ensure optimization.

Game should only allow one instance of a game at a time.

Game should require unique team and game names, and notify players if selected name is already in use.

## [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 Game Service, Game, Team, and Player classes are all associated with one another in some form. Game Service can contain a Game, Games can contain one or many Teams, and Teams can contain multiple Players. Each of the latter three classes inherit from the Entity class. The ProgramDriver class contains the main method, and uses the SingletonTester class within it.

**"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** | Apple has discontinued its server operating system, Mac OS X Server, but does currently still support existing users of this OS.  This OS is not recommended, as Apple does plan on dropping support for the discontinued OS, and the required hardware to run it is comparatively more expensive than a typical PC. | Linux distributions are generally free to use. Linux is a popular choice due to its stability, security, and open-source nature. This OS is the most popular for servers and web-hosting and is by far the most cost-effective.  Linux does require a fairly high level of technical expertise to use. | Windows is a popular choice due to its familiarity to most users and its ease of use, and its intuitive GUI. It is a good choice for applications that use Windows-specific technology such as .NET.  Windows requires a license fee for each server, which can be pricey when compared to other OS. Licensing costs can vary depending on the version, but range between $500-6,150 | Android mobile devices can be used to host web pages and apps. This is a cheap method for hosting and does not require a great deal of technical ability, but the capabilities of hosting this way are limited. Due to hardware restrictions in mobile devices, such as less RAM, performance using this type of OS may not be as stable or reliable as other methods. |
| **Client Side** | Mac devices run on languages like Objective-C and Swift, which are less common than some of their counterparts. This means specific expertise is likely to be needed, and additional time may be required to complete for Mac clients. | Linux supports a large number of languages, including some of the most popular programming languages like Java, Python, and C/C++. This offers a great deal of versatility, and means that specific expertise is not necessary. Developing for Linux could be done in a small team and likely completed quickly. | Similar to Linux, Windows also has a large array of programming languages with which one can develop. Windows also has its own specific languages and frameworks, such as C# and .NET or ASP.NET. Most users that access the website are likely to be using Windows OS, as it is the most common. | The game has already been developed for Android, but to introduce it to iOS would be similar to Mac development; Swift would be a necessity for an iOS mobile application. |
| **Development Tools** | Apple development is done via Xcode, which is a free IDE offered to Apple users. The drawback is that Apple development must be done on a Mac device, meaning costs may be incurred for the purchase of Macs, or the rental of virtual machines. | Development for Linux may vary depending on which language is used. Many IDEs or text editors are Linux-compatible, and many are free, and some have free versions available, with paid versions (costing up to $250/year per user) with extra features. | Visual Studio is one of the most commonly used IDEs for Windows development. It supports a large array of languages and is free to use. | Android development can be done using a free development tool from Google called Android Studio. Similar to MacOS development, iOS must be done on an Apple device with Xcode. |

## 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**: The recommended Operating System for the server is Linux. Linux will significantly reduce operating costs, and generally is more secure than any other platform available. It is the most popular platform for servers and is well-suited for this purpose.
2. **Operating Systems Architectures**: A server-client model is recommended, using our Linux server as a backend for game management, and utilizing client-side rendering using the end-user’s machine. Client-side rendering will reduce some of the more intensive processing on the server, retaining CPU power for essential game processing, and also reducing data center costs with reduced processing need.
3. **Storage Management**: Physical storage options include HDD or SSD, of which the latter is recommended. The SSD, compared to the HDD, allows for faster access of objects stored on disk space, which can help allow for smoother, faster gameplay.

Cloud storage is also an option with services such as AWS, but this is not recommended as an SSD should be sufficient to meet our performance needs, and is typically a one-time cost instead of a recurring subscription for cloud storage.

1. **Memory Management**: Memory usage can be managed with Linux’s page cache concept. Images to be rendered can be added to the cache, and stored there indefinitely (as long as space is available) to avoid needing to reaccess from disk storage. Memory space can be maximized by ensuring all images are properly formatted and compressed when possible.
2. **Distributed Systems and Networks**: With a server-client model, communication between front and backends will rely heavily on RESTful APIs. This communication will be dependent on reliable, high-speed internet connectivity on both the server and client sides. To minimize latency, ideally one server will be located within each major geographic region in which there is expected to be a high volume of players.
3. **Security**: Role based access is recommended to limit user access to unnecessary information or game assets. The principle of least-privilege access should be employed, ensuring, for example, that only team captains can edit any team-specific information such as the team name. No end user should have admin access.