

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 3-4**

**Evaluation 4-8**

**Recommendations 9-10**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/22/2024 | Emerald Tresch | * Recommendations for The Gaming Room Software * Update existing classes and add Entity Class * Updated Evaluation and Recommendations * Completed Recommendations |
|  | 10/06/2024 | Emerald Tresch | * Updated recommendations and system architecture view. |
|  | 10/20/2024 | Emerald Tresch | * Recommendations updated further |

**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 client seeks to revamp their game, "Draw It or Lose It," which is currently only available as an Android app, into a cross-platform web-based game. They require assistance establishing the development environment and optimizing the development process based on the existing Android application. The game should be created using a cross-platform framework. It's important that each game and team name be unique and that the game only allows one instance to exist in memory at a time. Additionally, each team will consist of multiple players, and the game should support multiple single teams playing simultaneously,

## Requirements

The client's requirements dictate that the game must incorporate one or more teams, each with multiple assigned players, and that only one game instance should be active at a time. This can be achieved by implementing unique identifiers for each game instance, team, and player. Furthermore, the user must be able to verify name availability when selecting a team name, needing the enforcement of uniqueness for each game and team name.

## [Design Constraints](#_2et92p0)

Each game must support one or more teams, each including multiple players. Game and team names must be unique to enable users to verify name availability. Additionally, only one game instance should be allowed to run simultaneously. The game must be compatible with multiple platforms and should be developed using Java and the Framework API.

Developers must adhere to these design constraints to ensure the game's proper functioning. Furthermore, the client's requirement that the game be run on multiple platforms poses a challenge, as the existing code is only compatible with Android devices. Adapting the code to other platforms will require additional development time. Other potential challenges include managing cloud space for web application development and addressing potential limitations on the number of games, teams, and players due to server constraints.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: Nothing is 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 UML diagram consists of seven classes. Game, Team, and Player have a generalization relationship with the general class Entity, demonstrating multiple inheritance. These three classes and the GameService class are directly associated and can be grouped, indicating that they may share zero to many objects. Multiple players can be added to a team, each identified by an ID and Name. Similarly, various teams can be added to a game, each identified by a name. The GameService class has a list of games and a singleton method called to service. This class also has references from the game class. Additionally, a SingletonTester class tests if a single occurrence of the game runs at a time associated with the ProgramDriver class. The primary method for the terms of use is stored within the ProgramDriver class.

**"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)

When considering potential development targets, evaluating platforms such as Macintosh, Windows, Linux, and various mobile platforms is essential for their suitability in hosting/serving or acting as the client. A comprehensive analysis of each platform's strengths and weaknesses is crucial for informed decision-making. It is important to emphasize that the server and client choices are not interdependent. For example, even if Linux is chosen as the server OS, Windows might still be the preferred option for the client. This decoupling of server and client choices allows for greater flexibility and optimization based on specific requirements.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | OS X Server is designed for Mac, but finding hosts may be challenging and costly unless the client is willing to buy their hardware. OS X Server costs $499 for up to ten clients or $999 for unlimited clients. | Due to its widespread popularity, Linux is the leading choice for web hosting operating systems. Thanks to its open-source nature, Linux boasts lower maintenance and licensing costs than closed operating systems such as Windows.  Typical cloud providers like Google and Amazon prefer Linux over Windows. | The appeal of Windows servers lies in their user-friendly GUI interface, which allows for seamless integration with the office's commonly used applications.  However, it's important to note that the licensing costs, often based on a per-user model, can be pretty substantial, particularly compared to Linux. Windows server licenses can range from $6,200 (for up to 16 core licenses) to $500 (for up to 50 clients) per installation per year.  Furthermore, it's worth mentioning that hosting platforms for Windows servers may not offer as much variety as those available for Linux. | Mobile devices can function as personal web or file servers but are not designed for multi-user access. Their hardware, such as RAM, is generally more limited and lacks the scalability of blade servers. Additionally, the costs associated with hosting tools for mobile devices are uncertain, as these tools would likely need to be developed in-house. |
| **Client Side** | To develop for Macs, you need a Mac computer with the latest version of Xcode.  Additionally, it is  important to note that the macOS SDK primarily uses Objective-C or SWIFT, which are not as widely recognized as other programming languages.  Finally, it is worth considering that Windows holds a 75% market share compared to macOS's 16%, indicating a smaller market opportunity for Mac development. | Developing in Linux should be relatively simple, as popular languages like Java, C/C++, or Python can be used. Additionally, the GNU/Linux platform supports multi-user functionality. However, some may argue that GNU/Linux development has limited value due to its relatively low usage. | Developing for Windows typically involves using C# or NET, widely used languages. Thus, creating a Windows client application is not an obstacle.  Since Windows XP, Windows has been a native multi-user platform.  With 75% of computer users preferring Windows as their operating system, it presents a strong business case. | Mobile devices are not designed for multi-user use. However, creating a client application for Android or iOS is straightforward.  The Android SDK is Java-based, so code developed for Windows and Linux might be able to act as a starting point.  iOS is SWIFT-based, so Mac's requirements apply, including hardware needs. |
| **Development Tools** | Mac developers use Objective-C and SWIFT as development languages. XCode is the standard IDE for Mac development, costing $99 per year per developer.  Insert  Retry | Linux development can be done using C/C++, Java, or Python. Python IDEs, such as NotePad++ and PyCharm, are often free. Several C/C++ IDEs are available for Linux, with Eclipse being a popular and free option that supports all these languages. | The Windows operating system is primarily developed using C# and the .NET framework. Microsoft's Visual Studio is a widely used integrated development environment (IDE) that supports various plugins and integrations, including Jenkins and TestComplete.  Visual Studio costs $45 to $250 per user per year, depending on the included features. | The Android SDK is based on Java. The most popular IDE for Android development is Android Studio, which Google developed as the official tool and can be downloaded for free.  Objective-C and Swift are the main programming languages used in iOS development, and Xcode is the primary IDE used.  Xcode is available for $99 per year per developer. |

## Recommendations

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

1. **Operating Platform**: Irrespective of the chosen frontend software, it is advisable to opt for Linux-based servers. Linux servers offer cost savings on licenses and do not impose limitations on data center access, as with Windows servers. The frontend may remain agnostic to the backend, with the connection established through APIs. Linux is known for its robust security and operability. As the most prevalent server platform, it provides access to various tools, including security software. Given the agnostic nature of the front end, it can be coded in the preferred language for the specific platform, such as SWIFT for iOS, Java for Android, and .NET for Windows.
2. **Operating Systems Architectures**: The proposed architecture entails implementing a backend server responsible for managing the game environment and frontend rendering. Given that the Draw It or Lose It gameplay does not rely on real-time Twitch reaction, synchronous transmission between the front and back ends is not essential.

Employing a contemporary backend using containerized microservices with Kubernetes or Docker holds the potential for scalability. However, selecting a cloud provider is pivotal, as most providers offer proprietary tooling.

Opting for client-side rendering enables the offloading of resource-intensive tasks from the server, potentially resulting in substantial cost savings in monthly data center expenses. Moreover, this approach ensures seamless gameplay by allowing the client to cache a predetermined number of subsequent images ahead of active gameplay, guaranteeing an uninterrupted user experience.

Decisions are pending regarding whether the application will be browser-based or a Java application when executed on PCs or Macs. Developing and enabling a browser-based game via Progressive Web App (PWA) presents a workable approach.

1. **Storage Management**: Storing data can be efficiently managed using a cloud storage solution, especially for small file sizes. This allows for quick and easy expansion or modification of the existing image library without physical hardware. The Windows server operating system provides effective memory management, enabling users to organize files on their hard drives and monitor available storage space. Additionally, the flexibility to select a specific location for saving files or applications enhances accessibility. Using cloud storage contributes to faster load times and improved speed. If the Game Room does not intend to purchase its hardware, no decision must be made on the storage medium, whether HDD or SSD. HDD or SSD should meet the application's performance needs, especially with caching behavior and client-side rendering.
2. **Memory Management:** Linux employs the page cache concept to store data in virtual memory and utilizes demand paging, effectively reducing memory usage by loading only actively used pages into memory. Page replacement is accomplished using the Least Recently Used (LRU) algorithm.

The Android Runtime (ART) and Dalvik virtual machine utilize paging and memory mapping to manage memory efficiently. This approach ensures that any memory modified by an application, whether by allocating new objects or manipulating mapped pages, remains resident in RAM and cannot be paged out.

Initially, memory management in iOS operated on a non-ARC (Automatic Reference Counting) basis, necessitating the manual retention and release of objects. With the introduction of ARC support, Xcode now automates this process during compilation, eliminating the need for manual object management.

1. **Distributed Systems and Networks:** Considerations regarding uptime and the prevention of outages are driving the increasing adoption of cloud-native architectures for numerous applications. Cloud providers can replicate and shift services across diverse deployments, mitigating large-scale outages. The seamless communication between the front and back end occurs through RESTful APIs, facilitating asynchronous communication. This approach guarantees transparent client/server communication for the deployed front end, irrespective of the platform (Android, Windows, iOS), enhancing system flexibility and compatibility.
2. **Security:** The Windows server operating system provides user account control settings to safeguard data in transit. It incorporates features such as memory allocation and a private virtual address space for processes, ensuring that they cannot be accessed by other processes unless explicitly shared. The VPN service capabilities help protect client data and accounts from unauthorized access. Furthermore, the built-in anti-spyware feature in the Windows operating system prevents malware and viruses from infiltrating the system. Additional protection can be obtained by purchasing programs like McAfee and Norton.