

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.3

## Table of Contents

[**CS 230 Project Software Design** 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 | 01/24/2024 | Yancarlo Guzman | Prototype Software Design |
| 1.1 | 01/24/2024 | Yancarlo Guzman | Evaluation of the characteristics, advantages, and weaknesses of various platforms. |
| 1.2 | 02/08/2024 | Yancarlo Guzman | Finalize Proof Read |
| 1.3 | 02/20/2024 | Yancarlo Guzman | Complete Project 3 Requirements and finalize |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is expanding its portfolio with a web-based game, akin to its Android app "Draw It or Lose It." This multi-platform game supports multiple games, each accommodating various teams with unique names and numerous players. Additionally, the new game will feature a substantial library of stock drawings.

## Requirements

1. **Team Functionality:**

* The game must support the participation of one or more teams.

1. **Player Assignment:**

* Each team should have the capability to be assigned multiple players.

1. **Name Uniqueness:**

* Game and team names must be unique, enabling users to check name availability when selecting a team.

1. **Single Game Instance:**

* Only one instance of the game can exist in memory simultaneously.

1. **Unique Identifiers:**

* Unique identifiers must be generated for each instance of a game, team, or player to ensure distinctiveness and facilitate memory management.

## [Design Constraints](#_2et92p0)

The design constraints for developing the game application in a web-based distributed environment involve the necessity to accommodate different software development kits for Android and web platforms. Despite their varied creation methods, both platforms serve the same function. This diversity poses a challenge for ensuring seamless functionality across multiple platforms. Additionally, the requirement for unique team names is crucial to prevent naming conflicts and facilitate a user-friendly experience. These design constraints impact the development process, requiring careful consideration of platform-specific intricacies and robust mechanisms for name uniqueness.

## [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 is the main method, working with SingletonTester to check if there's a GameService instance. Entity is the parent class for Game, Team, and Player, and these classes inherit Entity's attributes. Players are not associated with teams, but teams can have players. Teams are not associated with games, but games can have teams. Games are not associated with GameService, but GameService can have games. GameService can only have one instance of each game simultaneously. Each team can only have one player at a 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](#_2o15spng8stw)

I've analyzed Linux, Mac, Windows, and mobile devices based on specific requirements. The results are presented with a focus on how everything needs to work together seamlessly for our client.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac can serve as a server despite expensive licensing, and a MacBook is often necessary for efficient development due to macOS integration with hardware. | Linux stands out as the optimal choice for web-based scenarios. Its popularity and free licensing make it a cost-effective and widely adopted solution for hosting web applications. | Windows is recognized for its top-notch security and user-friendly setup. However, it comes with a higher cost due to expensive licensing. | Mobile devices, while not as powerful as computers, offer versatility and portability for hosting web applications. They excel in wireless hosting scenarios but may face limitations in processing power and hardware for resource-intensive applications. |
| **Client Side** | Mac offers user-friendly SDKs, but development requires a MacBook, potentially increasing overall costs. It's a worthwhile investment for those with specific needs. | Supporting multiple clients on Linux is cost-intensive, primarily due to the need for Python expertise, leading to increased development time. Proficiency in Python is crucial for effective client-side implementation. | Supporting clients on Windows demands high expertise, impacting both cost and development time. Proficient use of Windows tools is crucial for effective client-side implementation. | Supporting clients on Mobile Devices requires expertise in app development, impacting both cost and development time. Understanding the distinct usage of mobile devices compared to computers is crucial for effective client-side implementation. |
| **Development Tools** | Building software for deployment on Mac involves relevant programming languages and tools, including iCode, which is a tool that can be utilized for learning programming on the Mac platform. | Building software for deployment on Linux involves relevant programming languages and tools, with Python being notable as it is already installed on most Linux devices. Additionally, Linux provides the flexibility to incorporate various other programming tools into the development process. | Building software for deployment on Windows involves relevant programming languages and tools, with Visual Studio being the standard and preferred choice for coding on the Windows platform. Windows provides compatibility for various programming languages, offering flexibility in development. | Building software for deployment for Android, requires use Java or Kotlin with Android Studio. For iPhones, Swift is used with Xcode, requiring a Mac for development. |

## Recommendations

I have thoroughly analyzed the characteristics and techniques specific to various systems architectures. Based on this analysis, I am prepared to make a recommendation to The Gaming Room. Specifically, I will address the following points:

1. **Operating Platform**: The optimal suggestion is Linux as it offers the most efficient server environment for hosting Draw It or Lose It.
2. **Operating Systems Architectures**: Linux stands out for its stability and security, making it suitable for diverse tasks. Additionally, it enables seamless separation of system and hardware requirements.
3. **Storage Management**: Both HDD and SSD storage options are viable for various storage needs. However, the recommendation leans towards SSD due to its advantages, including faster asset access and higher storage capacity. Specifically for gaming applications, NoSQL databases are recommended for managing game data and user information, offering flexibility and scalability in handling dynamic datasets associated with games.
4. **Memory Management**: Implementing a system load watcher is advisable to optimize costs effectively. This approach proves beneficial during periods of low usage, allowing for the reduction of required memory. Similarly, it facilitates the addition of memory when needed, ensuring efficient resource management.
5. **Distributed Systems and Networks**: Hosting the system in the cloud ensures continuous server operation, minimizing downtime. In case of server issues, the ability to shift nodes to another server or initiate an alternative server ensures uninterrupted service. This approach is advantageous for hosting all essential components of the game, providing an opportunity to create client-specific operating systems for accessing vital information efficiently.
6. **Security**: A role-based security system is recommended for enhanced functionality and distinct separation among game, team, player, and user roles. This ensures that users are restricted from accessing information beyond their designated roles, reinforcing security measures effectively.