

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 |
| --- | --- | --- | --- |
| 1.2 | 12/15/2024 | Jordan Bishop | Added Executive Summary, Design Constraints, Domain Model   * 1. Added Evaluation   2. 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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

The game application will need to keep track of each game in progress, the teams, and the players. To ensure consistency when tracking these objects, they will be managed by the GameService class, which will ensure that no more than one instance of a particular game, team, or player will be created inadvertently. Each of these objects will have a unique name and identifying number that cannot be duplicated. This design allows multiple teams with multiple players to participate in the game.

## 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.>*

* A game will have the ability to 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. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

* Server load: The program should be as lightweight as possible to keep server strain to a minimum
* Latency: The game will not function as intended if there is significant latency between a player’s actions and the server’s response
* Cross-platform compatibility: The game cannot be reliant on software exclusive to one platform or operating system
* Security: Data transmission must be encrypted to protect the users and servers from potential threats

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

The Entity class is the base class that Game, Team, and Player inherit from. It holds data and methods that are common between all three of the inheriting classes, including unique names and identifiers. The GameService class manages the Games, Teams, and Players actively playing the game. It can get information about games in progress and provides functions to move from one game to the next. The ProgramDriver class instantiates the application and currently uses the SingletonTester class to ensure that multiple instances of the same game aren’t being created.

**"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** | Advantages:  Good development environment for Apple products. Focus on security and user experience. Native tools including virtualization.  Disadvantages:  Expensive hardware. Not commonly used on server side. Inflexible for non-Apple device support. | Advantages:  Many distributions focused specifically on servers. Inexpensive, if not free. Highly efficient and customizable. Community support and stability.  Disadvantages:  Can be difficult for a user to learn. | Advantages:  Highly compatible with Microsoft systems and tools (MSSQL, Azure, Active Directory, etc.). Familiar UI and security features for most users.  Disadvantages:  Expensive licensing. More likely to be a target of malicious attacks. Higher hardware requirements than Linux. | Advantages:  Portability. Occasionally used as a micro-server for IoT devices.  Disadvantages:  Low computing resources. OS not designed for hosting a server. Limited tools available. Overall a bad choice outside of niche uses. |
| **Client Side** | Requires special tools (Xcode) only available on MacOS. Non-Mac developers need to learn a new set of tools. High hardware cost. | Free and open source dev tools available. Can be run on less expensive hardware. Can be difficult for beginners to troubleshoot and develop for. | Some free tools are available, but most enterprise software will come with licensing fees. Higher hardware cost than Linux. Familiar platform for most developers. | iOS requires Xcode and a dev license, but Android development should be free. Android apps can be tested cheaply, but iOS apps require expensive hardware. |
| **Development Tools** | Xcode, Swift, Objective-C | Many free IDEs available. Python, Java, C, C++ | Visual Studio, .NET, C#, C++ | Java/Kotlin for Android, Swift/Objective-C 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**: Ubuntu Server. Pricing for Ubuntu Server is flexible, ranging from free to $500/yr, depending on the size of the business. Linux servers are highly performative compared to other options, and are fully capable of hosting a web based game for the lowest cost and highest performance. The main drawback is that it can be an unfamiliar platform for some developers, but the benefits outweigh this problem.
2. **Operating Systems Architectures**: Ubuntu Server is based on the Linux kernel, which is monolithic and highly performative. It does not come with a GUI, but open source GUI options like GNOME and KDE are available if necessary. Linux generally uses the ext4 file system, but can interact with many other file types such as FAT or NTFS. Hardware support uses dynamic kernel modules, which loads drivers at runtime.
3. **Storage Management**: For Draw It or Lose It, a small onboard storage drive will be sufficient. Since the game uses less than 2 GB of total assets, paying for cloud storage is unnecessary. File sizes and transfer speeds can be further improved through image compression.
4. **Memory Management**: The “Lazy Loading” strategy is recommended for memory management. This will only require the client to load images that need to be displayed at the moment. Browser caching can also be used by the client to retain images that have already been loaded.
5. **Distributed Systems and Networks**: The client-server model allows server and client development to progress and scale independently. This is at the core of the cross-platform design strategy. The backend (Ubuntu Server) will handle user data, authentication, the REST API gateway, and game logic. If the game scales beyond a single server, IaaS cloud servers could add redundancy and prevent downtime, as well as provide some load balancing and geographical coverage to reduce latency.
6. **Security**: Disk encryption will be used to protect data on the server, and HTTPS data encryption will protect data in transit. The server will use a centralized authentication tool such as OAuth to authenticate users across all platforms. The server also uses Role-Based Access Control through REST to manage user privileges. The client applications for all platforms will implement input validation and secure APIs to protect against XSS or SQL injection attacks.