

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 | 11/7/24 | Sushmita Dhital | Update on the document. |
| 2.0 | 11/26/24 | Sushmita Dhital | Updates on Evaluation section. |
| 3.0 | 12/9/24 | Sushmita Dhital | Updates and complete. |

**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)

Our client Gaming Room wants to develop a game “Draw It or Lose It” and the software design problem involves creating a web-based game application. Since the game is only available in Android app, the client wants to expand it to multiple platforms through a web-based version. Making a Java application that sticks to object-oriented programming concepts is the suggested resolution for this. Classes like GameService, game, team, and player will be included in the design, where classes will work together to handle the logic of the game. To ensure that game and team names are unique, a single base class called Entity will supply shared properties like ID and name to all entities. To further ensure efficient handling of game states, only one instance of the game will be permitted in memory at any given moment.

## 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)

There are different design constraints for developing the game application in a web-based distributed environment which are as followings:

1. Cross platform accessibility:

One of the important requirements here is that users must be able to access the game online, and it must work on different platforms like android, iOS or windows. This requires creating the system to support multiple users, in different platforms together. It also needs to manage and take care of online related problems like lagging, network strength, and possible outages. To guarantee a secure environment, security factors like data protection and user authentication should also be used in the design.

1. Need of Unique Identifiers:

The program needs to make sure that player identification, team names, and game titles are all unique. Here, mechanisms must be used to verify and enforce originality in real-time and, prohibit identical entries. Due to the needs of establishing mechanisms, this requirement has an impact on both the backend functionality and the database schema architecture.

1. Use of Java:

Only Java-based frameworks and libraries will be used in the development of the game. This limitation provides multiple platform compatibility and enables integration with current Java tools. But on this other hand, it needs strict compliance to code standards, needs best Java practices, and the usage of Java-supported tools.

1. Only One Instance of Game in Memory:

As pointed out in the requirements, the rule that there must be only one instance of game in the memory at all times is important. And achieving this is not always easy and simple. In order to prevent conflicts and keep consistent game experiences, it is necessary to carefully design game states and instances.

The development strategy of the projects is greatly influenced by the design limitations. First, cross platform accessibility must be prioritized since it is the web-based distributed system. Need of Unique Identifiers makes sure that all the required names are unique. While Java development makes it easy for cross-platform support, on the other hand Java coding standards must be followed. Also, only one Instance of Game in Memory limitations introduce more complexities, but it prevents conflicts and keep consistent game experiences. It is important to take these factors into account to make sure that the finished game fulfills The Gaming Room's functional, and technical needs.

## [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 following UML diagram shows the model of the game application. To meet the software requirements, it is made up of multiple classes that are interconnected, such as GameService, Game, Team, and Player. And they are explained below:

1. Entity class (Base class):

Classes like Game, Team, and Player are derived from the Entity class. With properties like id, name, and toString(), it gives other classes a common structure. This shows Object-Oriented Principle, inheritance. Here, properties like ID(long) and name(string) are shared by the Game, Team, and Player classes and derive from the Entity class. Duplication is avoided, and game is consistence by doing this.

1. GameService Class:

It functions as the main service in responsible for monitoring all element of the game, including ways to add games, retrieve games by name or ID, and determine how many games there are overall. By using the Singleton pattern, this class makes sure that the Game Service is only used once in the entire game. Managing common resources, such as game data, during the entire game is made easier using this. To test this behavior, the SingletonTester class and ProgramDriver connects as shown in the diagram by <<uses>>.

1. Game class:

A single game is represented by this class. This class has a list of Team objects in it and inherits from Entity. The fact that the Game class has a collection of Team objects, indicating that a game is made up of one or more teams, makes the composition principle clear in this case.

1. Team Class:

It serves as a team's representative within the game. It contains a list of Player objects and inherits from Entity as well. The Team class follows the composition concept of Player objects, just as the Game class. This means that a team is made up of several players.

1. Player Class:

It serves as a representative of each player. This class provides methods to express the player as a string and inherits common properties like ID and name from the Entity class. It also inherits from the Entity 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)

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** | MacOS provides a stable, versatile and user-friendly environment for server-side applications, which is web hosting in our case. Developers who like Unix-based systems for its GUI have used and liked it widely.  Advantages:  Secure choice for web hosting since it has powerful security features from its Unix base. Has good user experience. Offers frequent software and hardware upgrades.  Weaknesses:  Smaller user base. Devices and license are typically more costly. Suitable web hosting software is more limited. | Linux is a very popular platform for web hosting mainly because of its scalability and its open-source nature. It is used a lot for server deployments.  Advantages:  It is open-source environment. Most affordable for us since it can be deployed with little or no licensing fees. Supports many kinds of server software. Highly secure platform for web hosting.  Weaknesses:  It may be difficult to find exclusive software compared to other platforms. May need more technical knowledge to manage. | Windows provides a very user-friendly environment for web hosting, especially for users already familiar with it. Wide range of development environments and apps are supported.  Advantages:  Supports a huge software library. Easy to manage servers without strong technical knowledge. It has large user base. Reliable platform for managing large apps, like our online game.  Weaknesses:  Vulnerable to viruses and security breaches. Licensing costs can be higher. | Mobiles (iOS/Android) are becoming more important for server-side applications, it has become more common in cloud-based services like ours. But hosting on mobile is less common than traditional platforms.  Advantages:  It gives portable server setups. Can reach a larger user base. Smaller-scale web hosting may be cheaper for us.  Weakness:  Lacks the strong security. May not be able to host large-scale, complicated web applications.  Needs app store and strict app store policies. |
| **Client Side** | Development may need moderate cost, particularly if Xcode or other programming tools are used. Licensing fees for development might also need being a member of the Apple Developer Program. It is compatible with recent web technologies like HTML5, CSS3, and JavaScript, so time investment is moderate. Creating cross-platform apps like ours requires a moderate level of skills. It is important to be familiar with Apple's Human Interface Guidelines and the basics of responsive web design. Its development is like Windows, but some testing is required to verify compatibility with its environments. | Development is cheaper than Mac or Windows because it is open source. No license cost is required. But testing and ensuring compatibility across multiple environments may cause in additional expenses. More time is needed for devlopment, as it supports wide range of desktop environments, extra testing is required to ensure that HTML interface functions well across all platforms. High level of expertise with web development and responsive design technologies is needed like Linux-specific quirk. It is cost-effective but time-consuming. | Developing has moderate costs here too, just like macOS. Licensing fees for development tools like Visual Studio can add cost, but many tools and compatibility with browsers like Chrome and Firefox makes development simple. Development time is short, as it has great support for modern web technologies and IDE. Less expertise is needed to create cross-platform apps like ours. It will be simple for developers who are familiar with web technologies like HTML, CSS, JavaScript, and responsive design. It has moderate costs and a low time investment. | Developing cost is medium, especially for iOS, it involves enrollment in the Apple Developer Program. Since Android is an open platform, developing apps for it is more affordable. Development time is higher. It takes more design and programming effort to create an HTML interface that adjusts to different screen sizes, like phone or tablet. To make sure the app functions on both iOS and Android, more testing is vital too. It takes a moderate to high level of knowledge, particularly when building responsive designs that work effectively with touch screens and different screen sizes. It needs more time, moderate to high expertise and variable cost. |
| **Development Tools** | Swift is best suited programming language for Mac. Common languages for frontend development, HTML, CSS, JavaScript is typically utilized, while Python, Ruby, and JavaScript are used for backend development. Xcode, Visual Studio Code or Sublime Text can be used for coding. Notepad++ and other tools are also useful. Development team needs to have more knowledge in macOS development for features that are specific to this platform. Most of the tools are free like Xcode, but some specific element might cost us. | Programming tools like Visual Studio, Vim, PyCharm, Eclipse, Notepad++ can be used. Common languages for frontend development are HTML, CSS, JavaScript, and backend are Python, PHP, or JavaScript. Development team here should be familiar with using open-source technologies. Expertise with Linux-based server systems could be required, but it helps scaling the development process using open-source. A lot of Linux tools are free and open source. | Visual Studios, VS Code, NetBeans, Eclipse are popular IDEs here. Common languages for backend development are Python, JavaScript, and .NET, and HTML, CSS, JavaScript are used for frontend development. Development teams should be knowledgeable in windows specific technology as well as general web development. Since developers can interact with many IDEs, it is flexible, but teams may need different skill sets. Most of the tools are free like Visual Studio, but some specific element might cost us. | Android typically uses Java, while iOS uses Swift. HTML, CSS, JavaScript are utilized for web-based components, and frameworks like React or Angular are used to create responsive UIs. IDEs like Xcode is the main tool for iOS, and Android Studio is used for Android development. Needs knowledge of web technologies and native mobile app development for smartphones to be sure that the app functions properly on all devices with different screen size and touch screen. For Android development, Android Studio is free. Xcode is also free for iOS, but subscription in the Apple Developer Program is needed. |

## 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**: Especially considering cross-platform compatibility, LINUX is a perfect operating system to expand "Draw It or Lose It." As I have mentioned earlier, it is open source, so it is affordable and reliable compared to other systems. It is highly secure, which is crucial for our game since it protects user data. The game can grow as needed because Linux is compatible with several platforms like web, desktop, and mobile and it also mixes with cloud hosting services easily. It makes managing and updating the game across various environments simpler.
2. **Operating Systems Architectures**: As LINUX allows n-tier or multi-tier architecture, which divides the system into layers, it can be used in our case. Because of its scalability and modular architecture, game can function independently across multiple platforms for its various components, such as the UI and backend processing. It improves system operations, data management, and game updates and maintenance can be done easily.
3. **Storage Management**: LINUX can work well for storage management with a mix of good hard drives, SSDs and cloud storage. Also, as the game progresses, efficient data retrieval and management can be done by using MySQL as a good database system to manage structured data such as user profiles, scores, and so on. LINUX 's better cloud integration enables smooth scaling as the game grows, while SSDs and hard drives provide quick, dependable local storage for game data. Cloud storage allows backups and provides flexibility to manage growing user data without affecting performance.
4. **Memory Management**: A key component of managing many games at once is resource utilization and good multitasking, and LINUX is good at it. Game processes can freely allocate memory for game elements dynamically because of LINUX's dynamic memory allocation feature, which distributes memory at runtime as needed. Its paging feature divides memory into pages and loads only the necessary pages into RAM, also makes memory allocation effective. So, by managing memory access and execution permissions it improves the overall performance.
5. **Distributed Systems and Networks**: As REST APIs can be used in LINUX, it is a good way to communicate between different platforms. It follows the client-server model, in which a client makes requests to a server, like web browser/mobile device, and the server provides the required game data in reply. Because every resource like player data has unique URL, clients can simply access and interact with distributed data on many platforms. By ensuring loose coupling between components, this method improves overall stability and scalability by allowing the game to continue to operate even in the time of system failures or connectivity problems.
6. **Security**: Strong security features like user authentication, encryption, and updates are provided by Linux, preserving the safety of user data on many platforms. Linux is open source, which enables quick security upgrades and peer review. To prevent unwanted access, it lets us create strong passwords using terminal, and multi-factor authentication (MFA) as well. The security of our user data is further protected by its frequent system updates and its strong firewall.