

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 4**](#_Toc115077323)

[**Domain Model 4**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 6**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/22/2024 | Abhilash Krishna Raj | Executive summary, requirements, design constraints, and domain model are completed |
| 2.0 | 10/6/2024 | Abhilash Krishna Raj | Evaluation table is completed |
| 3.0 | 10/19/2024 | Abhilash Krishna Raj | Recommendations is completed |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to turn their current game, Draw It or Lose It, into a web-based game that works on multiple platforms. The main challenge is making sure the game runs smoothly with unique names for games, teams, and players, and that only one copy of the game is running at a time. To solve this, we will use the Singleton Pattern to make sure there's just one game running, and each game, team, and player will have a unique ID enforced through the iterator pattern. This plan will keep everything organized, help avoid errors, and make it easier to grow the game in the future.

## Requirements

Business Requirements:

* Support multiple teams and players in a game
* Make sure game and team names are unique
* Players must be able to join teams and play the game

Technical Requirements:

* Only one instance of the game should exist at a time
* Unique IDs must be created for games, teams, and players
* The system should support web and mobile platforms and be easy to expand

## [Design Constraints](#_2et92p0)

Web-based Environment

* **Constraint**: The game must run on multiple platforms over the web
* **Implication**: The system needs to handle many users at once, keep things fast, and make sure the data is the same for everyone

Single Game Instance

* **Constraint**: Only one instance of the game should be active at any time
* **Implication**: Using the Singleton Pattern keeps the game consistent, but we must make sure it's safe when many users interact with the game at the same time

Unique Identifiers

* **Constraint**: Each game, team, and player needs a unique ID
* **Implication**: The system must use the iterator pattern to generate unique IDs to prevent duplicate names or IDs

Dynamic Syncing

* **Constraint**: Game data must update in real time for all users
* **Implication**: The server must handle quick updates efficiently to keep all users seeing the same game state

Scalability

* **Constraint**: The system must be able to support more users and new features in the future
* **Implication**: The design should allow easy growth without needing major changes

## [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 UML diagram shows how the main classes are connected to meet the game’s software requirements. The Entity class is the base for Game, Team, and Player, which all share features like id and name through inheritance. This keeps things consistent and avoids repeating code, making it easier to manage multiple games, teams, and players. The GameService class controls all games, teams, and players and uses the Singleton Pattern to make sure there is only one GameService instance, making sure that the game system is always organized and consistent.

The Game class represents a game and holds a list of teams, while the Team class manages multiple players. Each player, represented by the Player class, has a unique ID, fulfilling the requirement for unique identifiers. This design follows object-oriented principles like inheritance, which allows Game, Team, and Player to share common properties. Encapsulation helps make sure that each class hides its internal details and provides controlled access through methods like addTeam() and addPlayer(), making it easier to manage teams and players. The Singleton Pattern in GameService helps make sure that only one instance manages everything, helping to maintain efficiency and consistency across the system while fulfilling the software requirements.

**"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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS can host web-based software applications using macOS Server, but it’s less common. Even though it’s secure and easy to use, it has higher licensing costs and doesn't scale as easily compared to Linux, making it less ideal for The Gaming Room's needs of handling thousands of players. | Linux is great for web-based software because it has free, open-source options like Apache which are scalable. It requires more technical knowledge but has no licensing costs, making it a top choice for servers. This is a great option for scaling to thousands of users while keeping costs low. | Windows Server does integrate well with Microsoft tools like .NET for web-based software but comes with higher fees for licensing. While Windows is easier to use, it’s not the most economical or scalable solution compared to Linux for hosting web-based applications with large player bases. | Mobile devices are not suited for web-based software page hosting but can run lightweight server apps using cloud-based approaches. However, these solutions are not powerful enough to help support a full web-based game with thousands of players. |
| **Client Side** | Web-based software must be optimized for Safari, and this means focusing on cross-browser compatibility. Development costs may increase slightly for focusing on MacOS/Safari compatibility. However, if The Gaming Room wants a seamless experience for all users, it’s important to prioritize web compatibility for macOS. | Linux users usually use Chrome or Firefox. Web-based software must be tested on multiple distributions, but development isn’t that expensive because Linux is open source. The flexibility of Linux helps make sure that The Gaming Room can reach a large audience, but additional testing will be needed to ensure compatibility. | Must ensure compatibility with Edge, Chrome, and Firefox. Windows is widely used, so it’s crucial to support, but it may require more resources and budget to cover all its versions. However, with the popularity of Windows, a good user experience on this platform will be valuable for maximizing The Gaming Room's audience. | Must ensure game is responsive to mobile environment and works across various screen sizes and operating systems but this will add complexity to the development process. Frameworks like React Native or Flutter can streamline this, but it will still require additional testing to make sure performance and responsiveness on both iOS and Android devices. |
| **Development Tools** | Tools like Xcode and VS Code are used for Mac. Development team would need knowledge of macOS, and some tools, like Xcode, along with budget for licensing costs. The Gaming Room will need to assess cost of using MacOS development teams with the need to ensure the app runs smoothly on Apple devices. | Free tools like VS Code and Atom are common. Linux supports a lot of languages and frameworks, making it cheaper but requires more technical knowledge across multiple development teams. Linux is great for cheaper costs while still providing necessary abilities to meet The Gaming Room’s needs. | Development uses Visual Studio, which supports both .NET and web technologies. Costs might come with Visual Studio for licensing along multiple development teams needed. The Gaming Room needs to consider licensing costs and make sure development teams have expertise in Windows tools and environments. | Mobile development tools like React Native, Android Studio are free, while Xcode has licensing costs. Mobile development adds complexity which requires multiple teams for design and testing across various devices and platforms. This increases costs and time but is needed if The Gaming Room's wants to expand to a vast mobile game audience. |

## 

## 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**: Using a Linux server platform is recommended because Linux is known for its flexibility, stability, and security features. This is why Linux is great for hosting web-based software applications that can be scaled in the future. It is also open-source which reduces licensing costs for The Gaming Room and allows them to deploy multiple instances without paying for extra server fees. Linux also works well with cloud services and allows seamless integration with other systems and this makes it the best choice for an operating platform.
2. **Operating Systems Architectures**: Linux uses a modular kernel design where drivers, file systems, and network tools can be added or removed if needed. This flexibility makes it very simple to customize for different hardware and this would in turn help its ability to run on various environments. Linux's architecture also uses multi-threading which is important for handling multiple game instances at the same time. Each game instance can run as a separate process by using Linux's process scheduling to keep everything running smoothly but efficiently.
3. **Storage Management**: Cloud-based storage should be used to help store the large library of high-definition image files needed by the game while still providing scalable storage options. The system should also use indexed file systems that can quickly access these images based on game requests for fast retrieval times. Cloud storage is also very reliable across different distributed systems as they have automatic backups which prevent any loss of data.
4. **Memory Management**: Linux uses a lot of memory management techniques that can be useful for Draw It or Lose It’s game which includes paging, swapping, and caching. Paging helps the operating system to divide memory into small parts and only load up the necessary parts accordingly, and this reduces the total memory usage in the game. Swapping helps make sure that data that is used less is moved to disk storage temporarily, and this clears the RAM for tasks that need to be handled immediately. Caching helps keep commonly accessed data in memory for later use, and this helps speed up the game’s performance by reducing the need to constantly get common game files from the storage. These memory management techniques provided by Linux help make sure that the game can handle multiple players at the same time without any performance issues.
5. **Distributed Systems and Networks**: To help Draw It or Lose It to communicate between various platforms, a distributed system should be set up. Linux-based microservices should be used, where each service handles a specific task like rendering images, tracking game scores, or allowing players to join, etc. These services can communicate with each other and share data smoothly and efficiently through REST APIs over standard HTTP protocol. Also, hosting these microservices on cloud platforms allows for benefits like easier scaling, load balancing, and regional coverage, which all help keep connections fast between various devices. Using a content delivery network as well can also speed things up by storing and sending content from servers closer to the game user, and this would be able to reduce loading times and bandwidth use. A distributed system setup like this would make sure the game performs well and smoothly, even if there’s connectivity issues or outages.
6. **Security**: Security will be provided to Draw It or Lose It’s game through Linux’s strong security features like access control lists, encryption, and firewalls. To keep data safe when it’s being sent between platforms, encryption protocols should be implemented to help make sure things like user login info and game data are protected. Also, adding multi-factor authentication would give user accounts extra protection when logging in to help stop any unauthorized access. For the distributed system as a whole, firewalls and VPNs should be set up in order to keep communication between different parts of the game protected. This way, only services that are trusted by the system can access and share data.