

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 | 09/21/2025  10/2/2025  10/15/2025  10/26/2025 | Shonderia Pryor | This is the Complete Project Software Design Template   |  | | --- | |  | |

**Instructions**

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room should have a game application which is web-based and can serve various games, teams, and players. There should be no repetition of any of the items by its ID and name. The app should also be able to manage a lot of users simultaneously without the loss of track of the data.

To address this, the architecture employs Singleton GameService that ensures that a single game manager is operating within the memory. This prevents interpersonal friction and leaves all things under a single roof. The Iterator pattern is also utilized to test the existence of duplicate names during addition of new games, teams and players. Such decisions make the system remain dependable, scalable and unproblematic to maintain as it increases in size.

## Requirements

## [Design Constraints](#_2et92p0)

Distributed environment based on web: Must process many parallel requests, which means that thread safety and synchronization must be considered.

Uniqueness of names: The system should ensure there are no duplicate names within the game, team and players and this is guaranteed with the help of iterators in the GameService.

Single GameService: There can be only one instance of a GameService which is easy to manage though it may be a bottleneck without optimization.

Scalability: Design should be able to accommodate scalability without compromise to unique and performance requirements.

Conclusions: To meet these limitations, iterators to ensure uniqueness, synchronization to ensure concurrent environment, and performance sensitivity are to be used attentively to ensure the Singleton is not a performance bottleneck.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

UML diagram demonstrates the relationship between the various components of the app. The top level is the Entity class which contains the common features id and name. All the classes Game, Team, and Player are examples of inheritance since they inherit entity.

GameService is a Singleton that in turn builds and becomes the owner of all the games. Game Teams Game Teams can play in many Game Teams: Game Teams can include many Game Teams; each Game Teams has Team Players: Game Teams have Team Players. This arrangement demonstrates encapsulation, as every set of games, teams, and players is confidential and can be changed only with the help of special mechanisms.

Other principles of OOP in the diagram are:

Abstraction: Entity The common properties are defined to avoid repetition of properties by sub classes.

Polymorphism Subclasses such as Game, Team and Player are overridden by corresponding methods such as toString to represent their information.

Lastly, the ProgramDriver initiates the program and finally, the SingletonTester verifies that Singleton pattern is implemented as per. All in all, the diagram illustrates that the design patterns (Singleton and Iterator) coupled with OOP principles enable the app to be structured, reusable, and scaled with ease.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | The Mac computers are technically capable of hosting the web-based applications based on Apache or Nginx, but they are not frequently used in enterprise server settings. Prices of licenses are also more expensive than those of Linux, as the application of macOS is not legal without Apple hardware. This renders Mac not so useful in large scale hosting, but it is occasionally adopted in development or testing. | Linux is the most common web basedapplication hosting platform. It helps to use Apache, Nginx MySQL and a number ofother server tools at no cost. The cost of the operating system is not even licensed, and this makes it highly economical. The strengths of it lie predominantly in stability, scalability and security. Its disadvantage lies in the fact that it might be more technically difficult to set up than the Windows. | Windows Server is very favorable in supporting the hosting of applications, particularly when such a system uses Microsoft technologies such as .NET or SQL server. Administrators who are already conversant with windows find it easy to do so. It has, however,great licensing costs on the operating system as well as some server tools. High-traffic web servers can also be a little less efficient on performance, when using Linux. | Web-based applications are not hosted using mobile devices. They, however, are connected withapplications on servers. Because of this reason, mobile devices are not considered as a server-side. They only act as clients who use hosted applications |
| **Client Side** | The key to supporting Mac users is to make sure that the application is going to work with Safari and Chrome. These browsers must betested during development, and Apple systems are more expensive, and as a result, test hardware may be costlier. The time spent in development goes up by a few seconds since cross platform compatibility needs to be checked. | Linux desktop customers are usually using either Firefox or chrome to access applications. Development must ensure that the web interface functions across these browsers. The costs are also low because Linux is open-sourceand no cost is to be mentioned;developer skills have tobe utilized to test and validate compatibility. | The Windows user base consists of desktop users most so that compatibility with Edge and Chrome is a must-have. Servicing Windows clients also does not contribute to a higher cost of licensing among the users but disrupts testing time in determining the responsiveness and stability between browsers. | When developing mobile apps, one must make sure that the application is responsive and that it works with mobile browsers such as Safari (iOS) and Chrome (Android). Other mobile testing experience is required, and testing may take longer since there are numerous various screen sizes and OS versions. |
| **Development Tools** | In the case of Mac, the developers usually rely on Xcode (in the case of iOS) and general-purpose software such as IntelliJ IDEA, Eclipse, or VS Code. The Xcode is an open-source tool, but you need a macOS system, which is expensive. There is a need to have developers who have experience in the iOS/macOS environments. | Free software such as Eclipse, NetBeans or VS Code may be used to develop Linux, and popular web stacks, such as LAMP (Linux, Apache, MySQL, PHP). The prices are cheap because the tools and the OS are open-source. Experience in Linux server administration might be required in the teams. | Visual Studio is popular on Windows, particularly to develop with .NET. It is rather costly to license (unless with the Community edition), yet it is powerful and can be used well with Windows Server. This platform will be simpler todevelopers who are already used to windows environments | Whether it is Android or iOS, mobile development normally involves Android Studio and Xcode respectively. Both are free, although the Apple developer program has an annual fee. Test devices need to be cross-platform and occasionally even divided teams if native applications are being developed. Standards such as React, Angular, or plain HTML/CSS/JS will also be used in the case of a web-based responsive app. |

## 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**: The optimal operating system to scale Draw It or Lose It is a Linux based server platform, like Ubuntu LTS, Ubuntu-based server platforms, with containerizedservices running on Docker and Kubernetes. Such architecture offers a scaling/portability/stability of many computing environments. Through the implementation of the game on cloud resource (such as AWS, Azure, or Google Cloud), The Gaming Room will be able to benefit by using the automatic scaling, regional presence, and economical infrastructure management. Containers will guarantee consistent performance of the game in every environment, and Kubernetes will offer automated deployment, load balancing and self-healing capabilities, stipulating that Draw It or Lose It can grow seamlessly as players increase in number.
2. **Operating Systems Architectures**: The operating system structure of the suggested platform is founded on the architecture of a distributed, micro-service. The game has each of its fundamental services (matchmaking, game session management, authentication, and media services) in a separate container that speaks to each other via secure APIs. Kubernetes must synchronize these services and make sure that scaling and fault tolerance is done across multiple worker nodes. The architecture consists of a control plane to control the state of the system, worker nodes to execute application services, and ingress controllers to route and balance traffic of the players. Isolation of resources is also realized by Linux namespaces and cgroups features which make each serviceuse both memory and CPU in an optimal and secure way. This architecture facilitates constant updates and reduced downtime which is very essential in ensuring a seamless gaming experience.
3. **Storage Management**: To store information, the platform ought to integrate different systems to maximize performance and cost. Large game assets like the drawings and images must be stored in an object storage service like Amazon S3 or Google Cloud Storage because it is scalable and inexpensive. The user profiles, leaderboards, and game session data of the database should be managed by a relational database (PostgreSQL) where consistency and transactions mattered. Also, Redis, an in-memory Data Store, can be deployed to retrieve fast changing data such as active sessions or a list of player presence to enhance response times during the gameplay. Cross-region replication and automated regular backups also need to be introduced to guarantee the durability of data and the rapid recovery of data in the case of failure. This mixed strategy enables The Gaming Room to be balanced in terms of reliability, speed and cost efficiency.
4. **Memory Management**: This platform manages the memory with both Linux kernel functionality and container-level management. Kubernetes and cgroups are used to allocate memory quotas to each containerized service to ensure that none of them uses large amounts of memory resources. The backend services which are created in languages like Node.js or Go can be optimised further with the help of runtime tuning and garbage collection control. Redis assists in unloading short-term data off the main memory enabling processes to be lightweight and efficient. The scaling of more containers by Kubernetes takes place automatically in response to either memory or CPU utilization, and the performance does not decrease, even when there is too much player activity. In other words, monitoring tools, such as Prometheus, should be an on-going process that will allow identifying memory leaks or an abnormal consumption of resources early enough to enhance the overall reliability of the system.
5. **Distributed Systems and Networks**: To facilitate the exchange of content across platforms, Draw It or Lose It can take advantage of distributed systems and networking architecture based on real-time communication protocols. Continuous, low-latency connections between the clients and the server should be maintained using Secure WebSocket so real -time drawing updates and interaction with gamers. Peer-to-peer communication can also be peer-to-peer enabled with WebRTC allowing further high-performance or regardless of the older client, long polling can be used as a fallback approach. To communicate with the backends, the lightweight protocol like GRPC or HTTP/2 can be used to interact with the microservices effectively. It ought to be implemented in various areas or zones with a process of automatic failover and replication to ensure its functionality even when the network is down. With the help of a Content Delivery Network (CDN), latency can be decreased by accessing the game resources using servers that are nearest to the game participants. The combination of these strategies leads to the development of a scalable network that is reliable and can serve users on various devices and operating systems.
6. **Security**: Since security is a priority of The Gaming Room, the suggested platform is based on a multi-layered security approach. Network communication should all be encrypted by TLS 1.3 and data stored in databases or object stores should be encrypted at rest by key management systems. Authentication and authorization are done using safe processes such as OAuth 2.0 and OpenID connect that ensures that the user is authenticated safely. Role based access control (RBAC) permits the application and Kubernetes environment. Input validation, to ward off injection attacks, rate limiting, to ward off denial-of-service (DoS) attempts and automated dependency scanning, to identify vulnerabilities in the codebase, should also be included in the system. Network security is further enhanced with the help of a Web Application Firewall (WAF) protection and DDoS. Privacy compliance should be attained through the reduction of personal data and the ability of the users to destroy their information. Daily auditing, tracking and incident response planning will help in ensuring that the information of the users is safe on all the platforms that are interconnected.