

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 09/19/24 | Ryan Orton | Updated Summary, Requirements, Constraints, and domain model |
| 1.1 | 10/06/1014 | Ryan Orton | Updated Evaluation |
| 1.2 | 10/17/2024 | Ryan Orton | Updated Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to expand their game, "Draw It or Lose It," to a web-based, multi-platform application from its current Android-only version. Your proposed solution involves utilizing software design patterns to create a scalable, maintainable, and efficient application that meets the client's requirements. Highlight the importance of ensuring unique identifiers for game instances, teams, and players, and the necessity of implementing both the Singleton and Iterator patterns to manage these aspects.

## Requirements

The Gaming Room has outlined several crucial business and technical requirements for the development of their game application:

1. The application must support one or more teams, each consisting of multiple players.
2. Game and team names must be unique, with a system in place to prevent duplication.
3. The game must enforce the constraint that only one instance of the game exists in memory at any given time.
4. Unique identifiers must be assigned to each instance of a game, team, or player to ensure consistency and prevent conflicts.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based, distributed environment introduces several design constraints:

1. **Performance and Scalability**: The application must efficiently manage concurrent users across multiple platforms, necessitating a scalable architecture that can handle varying loads without compromising performance.
2. **Network Latency**: Given the distributed nature of the application, network latency could impact the user experience. This requires careful design to minimize delays, particularly in time-sensitive game actions.
3. **Security**: Protecting user data across different platforms is paramount, especially in a distributed environment where data is transmitted over networks. Robust encryption and secure authentication mechanisms are essential.
4. **Cross-Platform Compatibility**: The application must be compatible with various operating systems and devices, requiring careful selection of technologies and design patterns that support portability and consistent behavior across platforms.

## [Domain Model](#_8h2ehzxfam4o)

The Entity Class is the parent (super) class of the Game, Team, and Player classes. This means that Game, Team, and Player class, as Entity’s child classes, will inherit Entity’s attributes, while each being assigned attributes of their own, that are separate to the parent class. The Game Service Class is used to ensure the client’s requirements are met, providing a single game instance at a time, unique team name (id), unique game name (id), and unique player name (id). Program Driver contains the main statement and uses the Singleton Tester class. The Game class contains a team list, Team class contains a Player list. The Player class does not contain a list, as it ensures that each player has a unique id, that can be assigned to a team. While a player can be on a team, and a team does have players, the player class does not contain or have a team or a game.

**"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** | Mac is a viable option for server-side development, though it is less common in production environments. It offers strong development tools and a Unix-based architecture, similar to Linux, but may require additional configuration for optimal performance. | Linux is an ideal choice for server-side deployment due to its stability, performance, and widespread adoption in web servers. It offers robust security features, a large support community, and cost-effectiveness, making it well-suited for hosting the game application. | Windows is commonly used in enterprise environments and offers good support for server-side development, but it can be resource-heavy and costly. It is less favored for web servers compared to Linux, though it remains a solid option depending on the existing infrastructure. | Mobile devices require a lightweight, responsive server-side architecture that can handle variable network conditions. The game server must be optimized for performance on mobile networks, which may include using mobile-specific protocols or frameworks. |
| **Client Side** | Mac offers strong support for client-side development with a focus on quality and user experience. Development costs and expertise required can be high, particularly for cross-platform support. | Linux is less common for client-side applications, particularly in consumer environments, but it can be efficient for certain types of client-side processing. Development tools for Linux client-side applications are available, but they might require specialized expertise. | Windows is widely used for client-side applications, particularly in corporate environments. It offers extensive support for software development, though the cost and complexity of developing for multiple client types can be significant. | Developing for mobile devices involves considerations of varying screen sizes, touch interfaces, and platform-specific requirements (iOS vs. Android). Development costs can be high, and expertise in mobile development is crucial for success. |
| **Development Tools** | Mac supports a variety of programming languages and tools such as Xcode, Swift, Java, and cross-platform tools like Flutter for deploying on Mac and iOS. | Linux supports languages like Java, Python, C++, and development tools such as Eclipse, IntelliJ, and Android Studio for deploying server-side or cross-platform applications. | Windows supports a broad range of languages and tools, including C#, .NET, Java, and Visual Studio, making it versatile for various types of software development. | Mobile development primarily uses tools like Android Studio for Android development, Xcode for iOS, and cross-platform tools like React Native or Flutter to target both platforms. |

**Recommendations**

1. **Operating Platform**: Linux is recommended as the primary operating platform for the game application due to its stability, scalability, and widespread use in web servers. It provides the best balance of performance, cost-effectiveness, and support for web-based distributed environments.
2. **Operating Systems Architectures**: The microservices architecture on Linux is ideal for the game's requirements, allowing modular development, ease of maintenance, and scalability. This architecture supports the distribution of services across multiple platforms while ensuring robustness and fault tolerance.
3. **Storage Management**: A cloud-based storage solution such as Amazon S3 or Google Cloud Storage is recommended. These services offer scalability, high availability, and integrated security features, making them well-suited for handling the game’s data storage needs across multiple platforms.
4. **Memory Management**: Linux’s memory management techniques, including efficient paging and the use of swap space, are well-suited for the game's needs. These techniques ensure that the application can handle varying loads and resource demands without performance degradation.
5. **Distributed Systems and Networks**: Implementing RESTful APIs for communication between platforms is recommended. This approach allows seamless data exchange between the game server and clients, ensuring consistent gameplay across devices. Consideration of redundancy and fault tolerance in the network design will help mitigate connectivity issues.
6. **Security**: Security is a critical concern for The Gaming Room. Implement encryption (e.g., TLS/SSL) for data in transit, use secure authentication mechanisms (e.g., OAuth 2.0), and ensure that user data is stored securely. Adherence to industry standards like OWASP Top Ten will help protect the game application and its users from common vulnerabilities.