

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 | 02/21/2025 | Susan Lopez | Initial Document Creation |

**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)

The Gaming Room wants to develop a web-based version of “Draw It or Lose It”, expanding it from an Android-only application to other platforms. For this application a Singleton Pattern will be used to ensure that only one instance of *GameService* exists. An Iterator Pattern will ensure each game is unique. Using an Entity Class Inheritance will simplify the code by creating a shared base class. This should allow the game to scale and function across multiple platforms.

## Requirements

1. Available on multiple platforms
2. Support Multi-player gameplay
3. Unique team and names
4. Secure User Data

## [Design Constraints](#_2et92p0)

1. Web-based and must work on multiple platforms
   1. Windows, Mac, Linux, Mobile Applications Android and iOS
   2. Responsive web-based interface so that the app runs smoothly on multiple platforms
   3. UI must be compatible with different screen sizes and resolutions
2. Unique Game and Team Names
   1. Name Checking Logic must be efficient
   2. Every game, session, and player should have a unique identifier to prevent duplicates
3. Requires a Singleton Pattern
   1. Need single instance of *GameService* active at one time
   2. Ensure all players interact with same game state and prevents duplication of game sessions
   3. Must handle concurrent user requests
4. Scalable
   1. Must be able to handle multiple users concurrently
   2. Low latency is required
5. Security and Data Protection
   1. User authentication enforced using OAuth 2.0, JWT tokens, or MFA
   2. Encryption protocols (SSL/TLS) must be used to secure data transmission
   3. Should comply with GDPR and other data protection laws
6. Latency and Real Time Synchronization
   1. The game is time based so low latency is required
   2. Server should use WebSockets or real time database updates to keep players synchronized
   3. Game Events need to be immediately reflected on all connected clients

## [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)

*GameService* is used as a singleton that manages multiple *Game* objects in a zero-to-many relationship. *Game* has one or more *Team* instances. *Team* has multiple players instances. *Game, Team,* and *Player* inherit from *Entity* as expressed with the open arrow (inheritance) pointing from *Game, Team*, and *Player* to *Entity*. The *Entity* Base class ensures that all game-related objects share common attributes: *id* and *name*. Attributes in each class are private (-) with public getters (+) to control access. This is an example of Encapsulation. Each class extends *Entity*, meaning they inherit common behaviors but can override methods if needed (Polymorphism). The *Entity* class serves as the general blueprint.

**"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 can host applications using Apache or cloud-based solutions. MAC is known for security and performance but is lacking in other areas compared to Linux. Hardware costs may also be higher. | Linux is the most widely used due to stability and flexibility. It is scalable and supports many web technologies including Apache, mySQL, and Node.js. There is a learning curve for administrators unfamiliar with Linux commands. | Windows is widely used and user-friendly with a robust enterprise support. It integrates well with .NET and SQL Server. Licensing Fees can be higher and it requires frequent security updates. | Mobile Devices are not meant to function as servers due to limited processing power and storage. They often lack the stability and security needed for hosting web applications. These devices typically connect to web based servers. |
| **Client Side** | Developers need to have expertise in Swift and Objective-C, as well as Xcode. There are fewer developers with these skills when compared to other OS. For users already on iOS or Apple the user experience would be seamless. For non-iOS users UX may be less intuitive. | Linux developers require knowledge of Electron, Java, or web-based applications. User adoption is lower due to the items mentioned above and a more complex ecosystem. | Most widely used OS with applications that can be built using .NET, C#, and Java. Using different versions of windows could cause issues or challenges across users. | Expertise needed can vary but usually includes Android (Java) and iOS (Swift/Objective C). Flutter or React Native could cut down on development time and cost. Apps would need to be optimized for different screen sizes and hardware capabilities. |
| **Development Tools** | * Xcode * Java * Electron | * Java * LAMP | * .NET * Java * ASP.NET * PHP * JS | * Android Studio * Xcode * Flutter * React Native |

## 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**: Linux-based cloud hosting with AWS, Google Cloud or Azure for the backend. The front end can be developed as a web-based application. A containerized environment will ensure consistency across platforms. Linux is also known for scalability and performance with no licensing fees. It also has better built-in security and less vulnerabilities than windows. It will also support various programming languages including Python, JavaScript, Java, and PHP.
2. **Operating Systems Architectures**: Backend will handle game logic, authentication, and data storage. Client applications will communicate with backend using APIs. Linux does a good job here because of the following characteristics:
   1. Monolithic Kernel
   2. Pre-emptive multitasking model
   3. Virtual file system
   4. Network stacking: supports TCP/IP, WebSockets, and RESTful APIs
   5. Microservices Architecture
3. **Storage Management**: A relational database like SQL could be used with Google Cloud Storage. Cloud Storage is a scalable, redundant storage for game images, player data, and logs. A relational database such as MySQL can store structured data such as users, game sessions, and leaderboards. A CDN can distribute game assests reducing latency and server load.
4. **Memory Management**: Linux uses caching and swapping. Many backend languages like Java use garbage collection and client devices use device-specific memory.
5. **Distributed Systems and Networks**: The backend server handles logic and game state, and clients will connect over a network. APIs will handle client-server communication. Swap space can be configured on Linux Servers to handle memory overflow. One can deallocate memory when assets are no longer in use. Using a Microservices architecture can divide game logic into small, independent services that communicate via APIs which allow for scalability and some fault tolerance. Using RESTful APIs & WebSockets will enable real-time multi-player communication.
6. **Security**: Multifactor Authentication could be used to enhance security, as well as OAuth 2.0. Data encryption and network security will be done on secure software and hardware while ensuring compliance with industry standards of security and applicable laws. Role-based access can also be used to restrict access to some features.