

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  1.2  1.3 | <07-18-2025>  <08-01-2025>  <08-12-2025> | <Ryan Blackburn>  <Ryan Blackburn>  <Ryan Blackburn> | Initial Version  Operating System Evaluations for Project adherence to Client demands. |

**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 Client *The Gaming Room* currently has an Android-only game, Draw It or Lose It, which they are seeking to adapt into a web-based application. The goal of the project is for the game to become available across multiple platforms. I propose that we use software architecture that is scalable and modular, and that uses object-oriented programming principles as well as industry standard design patterns. We will design the app using a centralized singleton, ‘GameService’, to ensure that a single instance of the game runs at any one time. We will also design an inheritance architecture using the ‘Entity’ class in order to maximize reusability. The end result will be a application that is able to be adapted to any platform and is also secure and maintainable.

## 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)

Because the goal of this project is to create a web-based version of an already existing game, we will be taking an already restricted architecture and making changes that will make it distributable across different operating systems. Doing this without having to completely rewrite the code will be the first obstacle, and it will also limit us to using tools that are platform independent.

In the program itself, only one version of the game should exist in memory at one time. This will be accomplished using the Singleton software design pattern, which will ensure that the game runs reliably and without errors that could be caused by creating accidental duplicates or code that is left behind.

The names for games and teams will be unique so that users don’t choose names that are already taken. This means that the program will require a means for checking existing names whenever a new game or team is created.

Lastly, because the game is web-based, a network connection will be required in order to play the game. This increases the demand for the game to be lightweight, so that it is playable on weaker connections, and secure so that vulnerabilities cannot be exploited.

## [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 Domain Model for the project outlines the core classes that make up the application architecture, showing how they relate to one another in order to fulfill the project requirements.

The primary class of the design is the GameService class, which is built around the Singleton design Pattern which ensures that there is only one instance of the game running at a time, fulfilling the clients’ requirements. GameService contains a list of Game objects and includes the logic for assigning unique identifiers to each game, team, and player.

The game objects inherit from the base Entity class, which serves somewhat as a template. This represents the Object-Oriented inheritance principle, as the three child classes inherit properties, such as a id and name, and methods, such as getId() and getName(), from Entity. This design ensures code reuse and consistency across all platforms.

Every Game contains a list of Team objects, and every Team contains a list of Player objects. This represents a Composition Relationship between the classes, creating a clear hierarchy for the modeling of the classes as well as ensuring that each game, team and player has a unique name, fulfilling the clients’ requirements.

There are also two utility classes, SingletonTester and ProgramDriver. These classes test to ensure that the Singleton is working properly and also serve to Run the application. They are independent of the rest of the code and do not interact with the game itself but are still integral to the stability of the game.

This system demonstrates strong object-oriented principles, including inheritance, encapsulation and modularity, and aligns with the clients’ design requirements and software standards.

**"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** | The Apple Mac OS supports server based applications, but it is not used primarily for hosting. Mac is primarily for development and testing of internal issues. | Linux is known for its ability to host web-based applications, because it supports server based deployment and is compatible with a multitude of backend tools. Linux is also scalable, and it is low cost because most of its distros are free. | While windows does still support web-based hosting, on Windows server platforms, it is better suited for backend services, provided that the licensing costs are not a problem. | Mobile devices are not intended to act as web servers, so while apps are able to interact with content locally, mobile devices are not a strong match for hosting server-side applications. |
| **Client Side** | Users of Apple Mac OS access the application via a web browser, so it is essential that compatibility with major browsers like Safari and Chrome are optimized. Testing on the Mac OS should be included when developing. | Users on Linux will access through web-based means as well, because MacOS is just a Linux Distribution. As long as the app follows the standards and is compliant with browser based apps, it will work on any Linux distro. | Windows is widely used for client side hosting. Modern web browsers are compatible with web-based applications, with little to no interference. | Mobile users will access the application through a mobile compatible web browser. This browser must be able to adapt to screen sizes and resolutions, which are much smaller than standard laptop or monitor sizes. These compatibility issues will require testing to ensure usability. |
| **Development Tools** | Apple Mac OS supports development tools like IntellJ IDEa and Eclipse, which means that Java code and design patterns can be used for backend development. | Linux is also compatible with Eclipse and Java development backend tools. It is known for server side programming because of its command line adaptability. | Windows is also known to support IntellJ IDEA and Eclipse, allowing engineers to develop the backend and test REST endpoints. | When developing for mobile, additional tools will be needed generally. However for this project, since the mobile client is a web app accessed by the browser, no mobile-specific tools are required. |

## 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**: To maximize the efficiency of the deployment, I recommend that it be deployed on a Linux-based server platform, preferably one running Ubuntu Server LTS, and that it be hosted in a cloud-based environment such as Amazon Web Services or Microsoft Azure. Using Linux will ensure that the game servers are stable, with very little cost. They will also be compatible with web server technologies like Apache, which are ideal for hosting Java-based apps. Being cloud based also makes the project scalable, and can account for spikes in server load, availability, and incident recovery.
2. **Operating Systems Architectures**: The platform will be on client-server architecture, with a thin client model. The server side, which will be Linux, will handle the game logic, sessions, and operations which affect the database. Clients will be able to access from any of the supported interfaces like web browser, desktop and mobile device, where they will communicate via HTTPS requests to RESTful endpoints. This architecture of separating the internal logic from the presentation makes the system modular, easily maintainable and accessible across most platforms. Using cloud infrastructure allows the game to load into multiple simultaneous instances through virtualization, keeping it balanced and readily available.
3. **Storage Management**: I also recommend a cloud-based storage solution such as Amazon Web Services or Azure Blob Storage. These will be used for static files like images and sound files. Alongside that we should use a relational database service, AWS RDS or Azure Database, for structured game data like user profiles and game states. By separating the volatile and non-volatile data we maximize the efficient retrieval of game assets that are frequently accessed without risking static files.
4. **Memory Management**: The linux platform will utilize demand paging and virtual memory management to allocate RAM, ensuring that only the code that is required will be loaded into memory while the game is playing. Server-side applications will be able to utilize the Java virtual machine’s garbage collection function to reclaim unused memory speedily, which will prevent memory leaks. Frequently used data will be stored using caching layers in the memory, which will reduce data load and increase response times.
5. **Distributed Systems and Networks**: The application will utilize a distributed architecture that is supported by cloud services, allowing it to have multiple instances of the application connected at once using a load balancer. Game sessions and player data will be synchronized across the servers using a central shared database and in-memory caching. Communication between the servers and clients will use HTTPS with Websocket for real-time gameplay features. The content delivery system of the cloud provider will host static assets with less demand for memory, getting them to the users faster. To mitigate the impact of outages, the system will include redundancies and automated failovers.
6. **Security**: Security will be enforced on the Transport Layer via encryption of all traffic between the server and the clients. This will reduce the chances of interception at a minimal decrease in speed. The platform will use secure authentication protocols and input validation and sanitization to resist injection attacks. The server will also be equipped with server firewalls, role-based access control, and intrusion detection systems, to protect its resources. Passwords and other sensitive data will be hashed and encrypted using AES-256, ensuring that user identities, game progress, and personal data remain secure across all platforms.