

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 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

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

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/25/25 | Jordan Poston | Web-based application, first pass |
| 2.0 | 06/07/25 | Jordan Poston | Overview for web-based gaming application |
| 3.0 | 06/20/25 | Jordan Poston | Final evaluation and recommendation of OS and mobile devices |

**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 following is an overview of the deployment of our gaming application on multiple platforms and devices. Our scope for this deployment covered the most widely used operating systems for desktops and laptops (Windows, Mac, and Linux) as well as the most popular mobile operating systems (iOS, Android). We want this game to be available to as many users as we can sustain across each of the various platforms and devices.

## Requirements

1. A scalable, web-based app with a server-style configuration that is capable of hosting thousands of players across multiple platforms.
2. A responsive, modernized HTML UI that can be accessed via traditional web browsers.
3. Seamless communication with the back-end application on the server.
4. Web support for each of the various platforms and devices.

## [Design Constraints](#_2et92p0)

* The application will require rigorous security measures and penetration testing for data security.
* The app needs cross-platform compatibility testing to ensure consistent experience across various platforms.
* The app should have an engaging and easy-to-use UI.
* The app should be able to handle multiple requests concurrently to accommodate multiple teams.
* The app should be able to maintain consistent performance as the overall player count increases.

## [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 showcases the six classes (Entity, GameService, Game, Team, Player, and SingletonTester) as well as the ProgramDriver. These classes utilize several principles of object-oriented programming. Inheritance is shown via the Game and Team subclasses inheriting various traits from the Entity parent-class. This use of inheritance cuts down on some redundant code. The classes in the model also utilize encapsulation by using both public methods and private attributes. The Singleton pattern is initiated by the GameService class via a self-referencing attribute. This practice guarantees that only one instance of this class exists in the app.

**"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** | Stable hosting environments and a very intuitive UI. Limited compatibility and licensing costs could be an issue when scaling for large amounts of players. | Stable and cost-effective server capacities make Linux strong for hosting. Open-source availability and free customization options make it ideal for scaling for large amounts of players. | Widely used option for web hosting. Windows has a large user base and boasts a large array of tools for developers. Windows can scale for large amounts of players but licensing costs for servers could be an issue for some. | HTML UI should grant compatibility across devices, guaranteeing consistent experience. Mobile platforms are not suited to host sessions for the gaming application. |
| **Client Side** | HTML interface would allow the use of default browsers for Mac users. This should keep costs low. | Linux users could also use their preferred bowsers, which keeps costs low. | As with Mac and Linux, Windows users can also use preferred browsers, keeping costs low for clients. | The HTML interface is ideal for a mobile deployment for both iOS and Android. The programming may need to be performed by dev team that specializes in mobile app development, to ensure a reliable experience on both mobile platforms. |
| **Development Tools** | Common Languages: Python, Java, Javascript.  IDEs : Eclipse, IntelliJ IDEA. | Common Languages: Python, Java, Javascript.  IDEs : Eclipse, IntelliJ IDEA. | Common Languages: Python, Java, Javascript.  IDEs : Eclipse, Visual Studio, ect. | For iOS: Swift is common with Xcode being the prevalent IDE.  For Android: Java is common with Android Studio being the preferred IDE. |

## 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 scale and cost-efficiency, a cloud-based architecture running a Linux OS would be my recommendation. This would reduce costs on maintenance, hosting and storage by paying for these services as a pay-per-use package with a company offering serverless solutions. This will also reduce stress on The Gaming Room as we move to scale. This solution also offers better support across the various environments that the application will be distributed on.
2. **Operating Systems Architectures**: Three-tier architecture includes a middle layer that passes information and executes logic between the user view and the data store. Serverless architecture modulates this layer, allowing them to offer expanded services in a neat package.

The Linux OS offers a lightweight solution that uses less disk space by utilizing a modular construction like the three

-tier architecture. The kernel contains the core components and acts as a bridge between the hardware and software components, while the separate modules encourage the dynamic expansion of services. A modular OS also avoids some latent communication via its layered OS.

1. **Storage Management**: Serverless solutions utilize a storage management layer that uses multiple servers and other technologies to create a storage environment for various media using a common interface. This leads to resource pooling, which can create greater redundancy and virtualization space.

Direct access storage is best overall for virtualization, UX, and load times. It uses indexed allocation schemes to cut down on unnecessary I/O data reads and eliminate wasted storage space due to fragmentation.

1. **Memory Management**: Linux uses virtual memory and demand paging to utilize partial memory execution. This allows multiple programs to run simultaneously, removes size constraints for physical memory, and loads/runs the application faster. Demand paging, however, does generate page faults that can affect performance if not managed properly.

Processes are stored on disk and are only pulled into memory when needed for execution. Linux uses page tables to map memory locations and track the pages that are currently pulled into memory. Linux also uses a Least Recently Used (LRU) algorithm to handle page swaps and reduce thrashing, which could otherwise adversely affect performance.

1. **Distributed Systems and Networks**: Serverless architectures are great for scaling. As the number of users increases, the application will have to run across multiple servers to handle the load. In a distributed system like this, load balancing becomes vitally important. Load balancing improves availability and response times by spreading client requests evenly across multiple servers. Distributed systems also address connectivity issues through redundancy. Traffic can be directed through one of many servers, avoiding downtime during maintenance.

Distributed systems often use master-slave replication to improve speed. This solution requires constant synchronization to utilize its single write node and its many read-only nodes. This solution streamlines the replication process as the process of actually writing data is far less frequent than retrieving read-only data. When a change is made in the master instance, the change is available for retrieval by the slave nodes immediately.

For serverless architecture, REST API using HTTP methods is a common setup that allows clients from disparate parts of the world to receive uniform service and communication.

1. **Security**: As previously stated, REST will provide stateless interaction and keep communication between the client and server lightweight. Sensitive data will be stored in the database, unavailable in any plaintext view. Additional security measures are triggered whenever the database is accessed in an unauthorized way. Browser security and transmission encryption are vital to ensuring the safety of session data and avoiding session hijacking.

Login credentialing (authentication and authorization) is the best fit for individual security for this application. Role-based access control is appropriate as this would allow users to construct teams and play the game while making no changes to the actual application.

Lastly, Linux is an excellent choice of OS as its robust security keeps each of its processes independent from each other. Keeping these processes independent also protects their respective memory areas and prevents rogue applications from overwriting or stealing data from other applications.