

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 | 05/31/2025 | Duncan Murchison | In this revision I will be editing the Executive Summary and the Design Constraints sections. |
| 2.0 | 06/25/2025 | Duncan Murchison | In this revision I will be editing the System Architecture sections. |
| 3.0 | 06/30/2025 | Duncan Murchison | In this revision, I will make my official recommendations for the System Architecture. |

**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)

In order to support multiple teams and ensure unique names for games and teams. The proposed solution uses an object-oriented, singleton based architecture to guarantee only one instance of the game in memory at a time. This will provide a maintainable foundation for future development.

## Requirements

Business Requirements:

1. Cross Platform Expansion: Redevelop the app to work with IOS, Linux, Mac, and Windows as well as Android.
2. Scalability: The app should be able to handle thousands of concurrent users, supporting both small and large-scale gaming events.
3. Cost Efficiency: Minimize costs for both server side and client deployments.
4. Security: Protect user data and system resources from unauthorized access or misuse.

Technical Requirements:

* Web Based Deployment: The core game should be browser based and accessible on both mobile and desktop.
* Server Side Hosting: The backend should be deployable on Linux, Mac and Windows servers.
* Browser Compatability: The frontend will support all major browsers (Chrome, Firefox, Safari, Edge)
* Resource Protection: Implement modern protection processes such as access control, least privilege, and domain separation.
* Development Tooling: Use programming languages and tools that enable maintainable and fast development with multi-platform deployment capabilities.
* Access Control: The system should enforce granular access policies for different user roles, including administrators and players.
* Audit Monitoring: Maintain audit trails for privileged operations to support monitoring and compliance.

## [Design Constraints](#_2et92p0)

* Platform-Agnostic Server Deployment: The server-side solution must be capable of running reliably and efficiently on Linux, Mac, and Windows systems, as well as being easily deployable on major cloud providers.
* Modern Web Standards for Client Compatibility: The client application must be developed using responsive web technologies (HTML5, CSS3, JavaScript, and a modern front-end framework). This ensures consistent user experience and functionality across all major browsers and devices
* Unified Codebase and Development Development Tools: The software architecture should emphasize a single, maintainable codebase for both server and client components, with widely adopted and cost effective tools and languages.
* Granular Access Control and Security Principles: The system must follow the principle of least privilege and domain separation, so that both system resources and user data are protected from unauthorized access or misuse.
* Audibility and Compliance: The platform should monitor for privileged actions and sensitive operations to support troubleshooting and compliance with industry best practices.

## [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)

Classes and Relationships:

* Entity: This is a base class that encapsulates common attributes (id, name) and behaviors (getId(), getName(), toString()) shared by Game, Team, and Player. By inheriting from Entity, these classes avoid code duplication and ensure consistent handling of identifiers and names.
* Game: Represents a single game session and contains a list of Team objects. Each Game can have multiple teams (0..\* multiplicity), supporting the requirement for team-based play.
* Team: Represents a group of players and contains a list of Player objects. Each Team can have multiple players (0..\* multiplicity), fulfilling the requirement for multi-player teams.
* Player: Represents an individual participant in the game and inherits from Entity.
* GameService: This class is responsible for creating and managing games, teams, and players. It uses the Singleton pattern (enforced by getInstance()) to ensure only one instance of the game exists in memory at a time, as required by the software constraints. GameService provides methods to add and retrieve games, teams, and players, and to enforce unique names.
* ProgramDriver and SingletonTester: Utility classes for running the program and testing the singleton implementation.

Object-Oriented Principles Demonstrated:

* Inheritance: The Entity class serves as a superclass for Game, Team, and Player, promoting code reuse and reducing redundancy.
* Encapsulation: Each class encapsulates its own data (such as lists of teams or players) and provides public methods to interact with that data, ensuring controlled access.
* Composition/Aggregation: The Game class contains (aggregates) a list of Team objects, and each Team contains a list of Player objects, representing "whole-part" relationships.
* Singleton Pattern: GameService uses the Singleton pattern to restrict instantiation to a single object, which helps enforce the requirement that only one game instance can exist in memory at a time.
* Polymorphism: By using the Entity base class, different entities can be managed in a generalized way where needed, making the codebase more flexible and maintainable.

The model meets all the requirements of multiple teams and players, the Entity base class and GameService methods enforce unique ID’s, as well as the single game instance.

"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** | Good for iOS and similar to linux, however lacks server specific features and its expensive. Not Recommended. | Open Source and containerization support, Requires high expertise but lightweight and has zero licensing fees. Highly Recommended | Significant licensing fees. However, its easy for windows based teams and there is frequent update support. Highly Recommended. | Not Applicable for server hosting capabilities. |
| **Client Side** | Since this is going to be a web application, testing for chrome, firefox, safari, and edge. Adapt to different screensizes, and use W3C standards for maximum compatability. | Since this is going to be a web application, testing for chrome, firefox, safari, and edge. Adapt to different screensizes, and use W3C standards for maximum compatability. | Since this is going to be a web application, testing for chrome, firefox, safari, and edge. Adapt to different screensizes, and use W3C standards for maximum compatability. | - Touch-friendly UI elements  - Optimize for mobile processors and memory  - Offline capability  - Push notifications via web APIs |
| **Development Tools** | Programming Languages:  - Swift  Dev Tools:  - Xcode  - VS Code  Database:  - PSQL  - MongoDB | Programming Languages:  - Java, Python Go, C#  Dev Tools  - Eclipse  - IntelliJ  - VS Code | Programming Languages:  - Java, Python Go, C#  Dev Tools  - Eclipse  - IntelliJ  - VS Code  - Azure DevOps | iOS:  - Xcode  - Swift/Objective-C  Android:  - Android Studio  - Java/Kotlin  - Google Play Console |

## 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**: For maximum scalability and cross platform compatibility, it is recommended that server side compoonents be deployed on a Linux operating platform. Using Docker for containerization and the ability to scale up with AWS or GCP. With tools like Docker and Kubernetes hosting modern web applications becomes a lot easier. There is also zero licensing costs with Linux operating systems.
2. **Operating Systems Architectures**: The Linux operating system is based on a modular, monolithic kernel architecture, providing efficient process management, memory management, and hardware abstraction. It supports both 32-bit and 64-bit architectures, though 64-bit is recommended for better performance and scalability. Linux's process isolation, user/group permissions, and pluggable security modules (like SELinux or AppArmor) make it suitable for secure, multi-tenant applications.
3. **Storage Management**: A hybrid storage solution is recommended, utilizing a relational database (such as PostgreSQL) for transactional data (user accounts, game sessions, scores) and an object storage system (such as Amazon S3 or MinIO) for large binary assets (drawings, images).
4. **Memory Management**: Linux uses advanced memory management techniques including virtual memory, paging, and demand paging, which allow applications to use more memory than physically available by swapping unused data to disk. The OS isolates the memory space of each process, protecting against leaks and unauthorized access. For containerized applications, (Like ones that run with Docker) the memory limits can be a useful tool for scaling techniques.
5. **Distributed Systems and Networks**: To enable seamless communication across platforms (web, mobile, desktop), Draw It or Lose It should be architected as a distributed system using RESTful APIs or WebSocket protocols for real-time interaction. The backend services can be deployed across multiple nodes (cloud VMs or containers), managed with orchestration tools (Kubernetes, Docker Swarm). For cross-platform connectivity, HTTPS ensures secure communication.
6. **Security**:

* Authentication & Authorization: Implement OAuth2 or OpenID Connect for secure login across platforms. Enforce role-based access control (RBAC) for administrators and players.
* Data Encryption: Use TLS (HTTPS) for all data in transit. Encrypt sensitive data in storage using built-in database encryption or file system encryption.
* Access Controls: Enforce least privilege for both users and services. Isolate containers and use network policies to restrict traffic only to necessary services.
* Audit Logging: Record all privileged operations, login attempts, and data modifications for compliance and incident response.
* Vulnerability Management: Regularly update system packages, dependencies, and monitor for vulnerabilities using automated tools.
* Cross-Platform User Protection: For mobile and client applications, secure local storage (e.g., Keychain for iOS, EncryptedSharedPreferences for Android), validate all inputs to prevent injection attacks, and minimize data retention on client devices.