

<Name-of-Software-Application>

# **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 | 01/24/25 | Andrew Lemus | <Brief description of changes in this revision> |

**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 game must support one or more teams, with each team consisting of multiple players. Both game and team names must be unique to allow users to verify name availability when choosing a team name. Additionally, only one instance of the game can exist in memory at any given time.

## Requirements

***Business Requirements***

1. *Enable gameplay involving one or more teams.*
2. *Allow each team to consist of multiple players.*
3. *Ensure unique game and team names to simplify name selection and prevent duplication.*

***Technical Requirements***

1. *Implement a system to verify name uniqueness during the naming process.*
2. *Ensure only one instance of the game exists in memory at any given time.*

## [Design Constraints](#_2et92p0)

1. Unique Naming System
   1. Constraint: The game and team names must be unique across the system
   2. Implications: Requires the implementation of a system to validate and enforce name uniqueness
2. Single Instance Enforcement
   1. Constraint: Only one instance of the game can exist in memory at any given time
   2. Implications: This may require a singleton design pattern to manage game state. Handling concurrency and ensuring only one active instance.
3. Scalability for Multiple Teams and Players
   1. Constraint: The system must support multiple teams, each with multiple players.
   2. Implications: The application must be designed to handle dynamic team and player management, including adding, removing, or updating teams and players without performance issues.

## [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)

**ProgramDriver:** Acts as the entry point for the program with a main() method. It initiates the application and interacts with the game service.

**SingletonTester:** Provides a testSingleton() method to ensure only one instance of the GameService class is created.

**Entity:** A base class for shared attributes across other classes

**GameService:** Manages the core functionality, including creating games, fetching games, and handling team and player IDs. Implements the Singleton pattern to ensure only on instance exists.

**Game:** Represents a specific game. Contains a list of teams and methods for adding or retrieving teams.

**Team:** Represents a team in the game. Contains a list of players and methods for adding players.

**Player:** Represents an individual player

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | - Limited use as a web server platform.  - macOS Server exists but lacks enterprise-level support.  - Expensive Mac hardware required.  - Not widely used for large-scale web hosting. | - Industry standard for web hosting.  - Supports Apache, Nginx, MySQL/PostgreSQL.  - Open-source and highly scalable.  - No licensing fees, reducing costs.  - Strong security and stability. | - Windows Server offers IIS (Internet Information Services).  - Can integrate with Microsoft technologies (ASP.NET, SQL Server).  - Licensing costs can be high.  - Requires more resources compared to Linux. | - Mobile devices do not host web applications directly.  - Apps rely on web-based APIs or cloud services for back-end functions.  - Deployment involves mobile app stores (Google Play, App Store). |
| **Client Side** | - Requires optimization for Safari and other major browsers.  - Development requires macOS for iOS compatibility.  - Expensive hardware requirements.  - Less gaming adaptation compared to Windows. | - Works with open-source web technologies (Chrome, Firefox).  - Requires cross-browser compatibility testing.  - Lower adoption among non-technical users.  - Cost-effective with free tools. | - Compatible with most web browsers (Chrome, Edge, Firefox).  - Development tools like Visual Studio are well-supported.  - Requires testing for Internet Explorer compatibility in some cases. | - Web apps must be responsive and mobile-friendly.  - Progressive Web Apps (PWAs) or native development needed.  - iOS apps require Mac hardware for development.  - Android development can be done on any OS. |
| **Development Tools** | - Xcode is required for iOS development.  - Supports Swift, Objective-C, JavaScript, and web frameworks.  - Can use IDEs like VS Code, IntelliJ, or WebStorm for web apps. | - Uses open-source tools: VS Code, Eclipse, Atom.  - Programming languages include Python, JavaScript, PHP, Java.  - MySQL/PostgreSQL for database management. | - Visual Studio is the primary development tool.  - Languages include C#, JavaScript, .NET, Python.  - Proprietary software may have licensing costs. | - Android Studio (Java/Kotlin) for Android.  - Xcode (Swift) for iOS.  - Cross-platform tools like React Native or Flutter reduce development time. |

## 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 expand Draw It or Lose It across multiple environments, a cross-platform framework such as Unity or Unreal Engine should be considered. These platforms support Windows, macOS, Linux, iOS, and Android, ensuring compatibility and ease of development.
2. **Operating Systems Architectures**: The recommended operating systems should support 64-bit architectures to handle large assets and ensure smooth performance. Windows, Linux, and macOS provide robust support for high-performance gaming applications, while mobile versions are optimized for ARM-base processors.
3. **Storage Management**: A combination of cloud-based storage and local caching should be implemented. Cloud storage solutions such as AWS S3, Google Cloud Storage, or Firebase allow for scalable image storage while reducing device storage requirements. Locally caching frequently used assets will improve performance and reduce latency.
4. **Memory Management**: The recommended platforms use memory management techniques such as:
   1. **Virtual Memory Management:** Ensuring efficient use of RAM by swapping data between physical memory and disk storage when needed.
   2. **Garbage Collection:** Automatic deallocation in managed environments like Android, iOS, and Unity’s C#
   3. **Efficient Asset Streaming:** Loading only necessary assets into memory to minimize footprint and avoid performance bottlenecks.
5. **Distributed Systems and Networks**: For seamless cross-platform communication, a client-server architecture should be employed. WebSockets or REST APIs can enable real-time data exchange between clients and servers. A cloud-based backend such as Firebase or AWS Lambda can handle game state synchronization, ensuring data consistency across platforms. Load balancing and redundancy mechanisms should be implemented to mitigate connectivity issues and outages.
6. **Security**: Security measures should include:
   1. **End-to-End Encryption (E2EE):** Protecting data transmitted between clients and servers using TLS/SSL
   2. **Authentication and Authorization:** Implementing OAuth, JWT, or multi-factor authentication (MFA) for user account security
   3. **Data Protection:** Encrypting stored user data and implementing role-based access controls (RBAC) to prevent unauthorized access
   4. **Regular Security Audits:** Conducting periodic security assessments to identify and mitigate vulnerabilities