

<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 | 06/01/25 | Dominick Griggs | Initial draft of software design document |

2.0 6/15/25 Dominick Griggs Minor improvements  
3.0 6/28/25 Dominick Griggs Recommendation improvements

**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)

## Creative Technology Solutions (CTS) has been contracted by The Gaming Room to redesign their existing Android-only game, Draw It or Lose It, into a web-based, multi-platform game application. The existing game is a drawing-based guessing game where players try to identify a clue based on a series of images. Our proposed solution involves developing a scalable, distributed game application with a strong foundation in object-oriented design and modern development patterns.

## This document outlines the software design approach to implement the required features: multi-team support, unique naming enforcement, and a single in-memory game instance. The solution will utilize industry-standard patterns such as Singleton (for centralized game management) and Iterator (to handle collections of teams and players). This design ensures code maintainability, data integrity, and readiness for future scaling.

## Requirements

*Business Requirements:*

*The game must allow multiple teams, each with multiple players.*

*Team and game names must be unique and validated upon creation.*

*The system should ensure only one active game instance exists in memory at a time.*

*The solution must be web-based and able to support mobile and desktop clients.*

*Technical Requirements:*

*Use of object-oriented programming principles.*

*Application of Singleton and Iterator design patterns.*

*Unique ID and name enforcement for game objects (Game, Team, Player).*

*Use of Java for the backend implementation.*

*Readiness for future cloud deployment and scalability.*

## [Design Constraints](#_2et92p0)

The game must be developed for a distributed, web-based environment, which introduces several constraints:

* **Concurrency Management:** The system must prevent duplicate instances, especially when multiple users attempt to start or join a game simultaneously. This requires proper synchronization and Singleton enforcement.
* **State Management:** Since web applications are inherently stateless, game state must be managed efficiently in memory or a backend service.
* **Name Uniqueness:** Validation of unique names must be quick and reliable. This adds overhead to operations like adding new teams or players.
* **Scalability:** While the initial application is simple, it must be designed to scale horizontally across servers and clients.
* **Cross-Platform Compatibility:** The design must consider varying behavior and performance across different client platforms (e.g., browsers, mobile devices).

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram defines four primary classes: Entity, Game, Team, and Player. Entity is a base class that encapsulates common attributes (id and name) shared by all other classes. Game extends Entity and contains a list of Team objects. Each Team (also derived from Entity) contains a list of Player objects, which also inherit from Entity.

Object-Oriented Principles Used:

Inheritance: Team, Player, and Game inherit common behavior from Entity, promoting code reuse.

Encapsulation: Each class hides its internal data and exposes only necessary methods for external access.

Abstraction: High-level classes like GameService provide an abstract interface for game management.

Design Patterns:

Singleton Pattern ensures only one instance of GameService exists.

Iterator Pattern supports traversal over game collections to check name uniqueness.

This structure allows us to maintain a clear hierarchy, enforce constraints (e.g., uniqueness), and build extensible components.

**"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 and secure; macOS servers are rare and less supported in enterprise environments. Good for development. | Linux is widely adopted for servers; offers robust security, scalability, and open-source tools. | Windows Server is viable but may involve higher licensing costs; less ideal for scalable web deployment. | Not suitable as a primary server host. Typically acts as a client. |
| **Client Side** | Minimal additional work; supports major browsers. Some QA needed for Safari. | Similar to Mac; lightweight browsers and strong development tools. | Broad compatibility; Microsoft Edge and Chrome widely used. | Requires responsive design and optimization for Android/iOS performance. |
| **Development Tools** | IntelliJ IDEA, Eclipse, Java SDK, Maven, Git, Docker, VS Code | Same tools as Mac; command-line tools more accessible. Ideal for continuous integration. | Visual Studio, Eclipse, JetBrains tools, Windows Subsystem for Linux (WSL) | Android Studio, Xcode (iOS), cross-platform frameworks like 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:

Operating Platform:  
CTS recommends deploying the server-side application on a Linux platform. Linux is the industry standard for hosting scalable, distributed applications due to its reliability, performance efficiency, and full compatibility with open-source tools such as Docker, Kubernetes, and CI/CD pipelines. Its cost-effectiveness and robust community support also make it an ideal choice for long-term maintenance and flexibility as the game scales.  
  
Operating System Architectures:  
Linux follows a monolithic kernel architecture, meaning all essential services (file system, process management, device drivers) operate in kernel space, which offers high performance and direct communication between modules. Its modularity also allows the customization of features to improve efficiency and security. The architecture supports virtualization, containerization, and optimized resource control—all of which are vital for a web-based, cloud-deployable game environment.  
  
Storage Management:  
For structured data such as player records, team names, and game statistics, CTS recommends a cloud-hosted relational database like PostgreSQL or MySQL. These platforms support ACID transactions, scalability, and high availability. For large binary assets such as drawing files or images, an object storage service like Amazon S3 or Google Cloud Storage can be integrated for cost-effective, scalable access. Caching tools such as Redis may also be introduced to reduce latency for frequently accessed data.  
  
Memory Management:  
The combination of Java and Linux creates an efficient memory management environment. Java’s garbage collection system manages heap memory allocation and cleanup automatically. Meanwhile, Linux enhances memory efficiency through advanced features such as demand paging, virtual memory management, and process isolation. The operating system can also dynamically adjust swap usage to protect performance under high load conditions, keeping gameplay smooth even during peak usage.  
  
Distributed Systems and Networks:  
Draw It or Lose It should be implemented as a RESTful web service to ensure interoperability across client platforms. The backend service should be containerized using Docker and orchestrated using Kubernetes to support horizontal scaling, high availability, and automated fault recovery. A cloud-based load balancer can distribute incoming traffic across replicas. These tools work together to ensure that service continues uninterrupted even in the case of hardware failure, network congestion, or service spikes.  
  
Security:  
Security must be prioritized at every level. HTTPS should be enforced to secure all data in transit. User authentication and session management should be handled via OAuth 2.0 to allow for secure login and token-based access. Sensitive data should be encrypted both in transit and at rest using TLS and AES-256 respectively. On the server level, Linux supports firewall configuration (e.g., UFW or iptables), user permissions, and SSH hardening to prevent unauthorized access. Additional layers of protection such as role-based access control (RBAC) and intrusion detection systems (IDS) can be implemented to further harden the environment.  
  
These recommendations are tailored to ensure that Draw It or Lose It operates as a secure, high-performance, and scalable platform that can effectively serve a growing user base across web and mobile platforms.