

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 | 08/17/2025 | Sharaine Pestano | redevelopment of popular mobile game, Draw it or Lose it with completed recommendations. |

**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 Gaming Room has requested the redevelopment of their popular mobile game, Draw It or Lose It, into a web-based version capable of supporting multiple platforms. The current application exists only on Android, limiting accessibility and scalability. To address this, Creative Technology Solutions is proposing a modular and distributed software architecture built using Java. The solution applies object-oriented design and proven software design patterns, such as Singleton and Iterator, to ensure memory efficiency, entity uniqueness, and centralized control. This approach not only satisfies the client’s immediate goals but also establishes a solid foundation for future expansion.

## Requirements

*Business Requirements:  
• Support for one or more teams per game  
• Each team must have multiple players  
• Game and team names must be unique  
• Only one instance of a game can exist in memory at a time*

*Technical Requirements:  
• Must be implemented in Java  
• Must support Singleton design pattern for centralized GameService  
• Must ensure unique identifiers for game, team, and player instances  
• Must use Iterator design pattern to enforce name uniqueness when adding entities*

## [Design Constraints](#_2et92p0)

The application must run in a distributed, web-based environment, creating the following constraints:  
1. Singleton Pattern Across Instances: Ensuring only one GameService instance exists in a distributed system can be challenging and may require synchronization strategies or server-level session management.  
2. Unique Naming and Identifiers: Game, team, and player names must be globally unique. Efficient name-checking mechanisms such as iteration through collections or future use of hashed structures must be considered.  
3. Cross-Platform Compatibility: The application must abstract system dependencies to allow deployment across different web browsers and mobile devices.

## [System Architecture View](#_ilbxbyevv6b6)

This section will describe the system and subsystem architecture for Draw It or Lose It. It includes a logical topology of components and communication layers for client-server interactions and real-time game synchronization.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram includes four main components: GameService, Game, ProgramDriver, and SingletonTester.  
• GameService: Manages the list of Game instances using a Singleton pattern. This ensures that only one instance of GameService exists in memory. It maintains a List<Game> and a counter nextGameId.  
• Game: Represents an individual game, storing its unique ID and name. It will inherit from a base class Entity, which includes shared fields id and name.  
• Entity (implied): A base class used for Game, Team, and Player to avoid redundancy in defining ID and name attributes.  
• ProgramDriver: Provides the main() method to run the program.  
• SingletonTester: Contains a method to test the Singleton implementation.  
OOP Principles Used:  
• Encapsulation: Private attributes with public accessors  
• Inheritance: Game, Team, and Player inherit from Entity  
• Polymorphism: Inherited classes override shared behaviors like toString()  
• Design Patterns: Singleton (for GameService), Iterator (to ensure unique naming)

**"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** | macOS offers a robust Unix-based foundation... | Linux is the most common platform for server-side deployment... | Windows Server supports .NET and Java-based server applications... | Mobile devices (iOS/Android) are not viable for server-side hosting... |
| **Client Side** | macOS supports modern web browsers and rich graphics rendering... | Linux users often utilize browsers like Firefox or Chromium... | Windows is the most widely used desktop OS... | Requires UI/UX optimization for touch interfaces... |
| **Development Tools** | Supports IDEs like IntelliJ, Eclipse, Visual Studio Code... | Open-source tools dominate... | Excellent IDE support: Visual Studio, Eclipse, IntelliJ... | Android Studio (for Android), Xcode (for iOS)... |

## Recommendations

**Operating Platform**

Linux is the most suitable operating platform for hosting the Draw It or Lose It application. It is the industry standard for server-side deployments due to its efficiency, scalability, and cost-effectiveness. Cloud providers (AWS, Azure, GCP) offer optimized Linux distributions with long-term support, and the platform integrates cleanly with containerization and orchestration technologies. Using Linux ensures that the application can be deployed in a modular and distributed fashion with minimal overhead.

**Operating Systems Architectures**

Linux architecture supports a microservices-based design, allowing each functional component (API services, real-time WebSocket gateway, storage services) to run independently in containers. Process isolation is achieved through namespaces and control groups (cgroups), ensuring each service runs with its own resource limits. The system architecture leverages Docker containers managed by Kubernetes, enabling autoscaling, load balancing, and health monitoring. Immutable container images provide consistent deployments, while service discovery allows seamless communication between subsystems.

**Storage Management**

The application requires both persistent and ephemeral storage systems. A **relational database** such as PostgreSQL (with unique indexes to enforce global uniqueness of games, teams, and players) will serve as the authoritative store. **Redis** will provide in-memory caching for active sessions, presence data, and real-time score updates. Game assets, such as drawing frames, will be stored in **object storage** (e.g., AWS S3) and delivered to clients through a CDN for high availability and performance. Log data and metrics will be centralized using a monitoring stack (e.g., ELK or Prometheus/Grafana). Backup strategies will include automated snapshots and cross-region replication for disaster recovery.

**Memory Management**

On Linux, the **Java Virtual Machine (JVM)** manages memory for the application. Garbage collection (G1GC or ZGC) ensures efficient cleanup of unused objects, minimizing pause times critical for real-time play. Heap memory is configured per container using JVM flags and Kubernetes resource limits to prevent memory contention or swapping. Thread pools are bounded to avoid over-allocation, and non-blocking I/O is leveraged for WebSocket connections. Redis provides off-heap memory management for ephemeral state, while object reuse and careful lifecycle management prevent leaks and performance bottlenecks.

**Distributed Systems and Networks**

Draw It or Lose It must operate across devices and platforms. RESTful APIs (over HTTPS) handle session management, account creation, and persistence of scores. Real-time interactions are supported by **WebSockets**, enabling synchronized clue reveals, guess submissions, and scoring updates. Redis pub/sub or a message queue (e.g., RabbitMQ, SQS) supports scalable fan-out of real-time events to clients. The distributed system design incorporates redundancy and failover: services are stateless and replicated across multiple nodes, ensuring that connectivity issues or outages in one zone do not disrupt the overall system. CDN integration ensures efficient asset distribution with minimal latency.

**Security**

Security is paramount in protecting user information. **Transport security** is achieved via TLS 1.2/1.3, ensuring encrypted communication across all platforms. Authentication is token-based (JWT or OAuth2), replacing basic authentication with short-lived access tokens and refresh tokens for improved safety. Role-based access control ensures users, moderators, and administrators have the least privilege necessary. Sensitive data such as passwords are salted and hashed using Argon2 or bcrypt. At-rest encryption protects database volumes and object storage. Input validation, request rate limiting, and WAF rules defend against injection, replay, or denial-of-service attacks. Logging and audit trails provide observability, while compliance features such as account deletion/export maintain user trust and regulatory readiness.