

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_kviby8w0752g)

[**Table of Contents 2**](#_j7ygf9hyycwc)

[**Document Revision History 2**](#_5li8yzkfzca3)

[**Executive Summary 3**](#_msplp6vx514b)

[**Requirements 3**](#_l0wp0peac1vm)

[**Design Constraints 3**](#_dtxqdofbz5ib)

[**System Architecture View 3**](#_xfui8me53vie)

[**Domain Model 3**](#_reh66byl3r68)

[**Evaluation 4**](#_m76nf37kfds7)

[**Recommendations 5**](#_ltv8b0i565uo)

## [Document Revision History](#_8ts4sf4h5jyn)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/30/25 | Sabrina Bitecofer | Initial draft of software design document |

**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](#_sard1l8hkc24)

The Gaming Room has requested a web-based version of their Android game *Draw It or Lose It*, which supports multiplayer gameplay where teams guess stock images as they are rendered. Creative Technology Solutions (CTS) proposes a design solution using object-oriented programming principles and standard software design patterns to ensure scalability, maintainability, and platform compatibility. The proposed solution ensures only one active game instance exists at a time, allows multiple teams with multiple players, and enforces name uniqueness using UUIDs and a central validation system.

## Requirements

**Business Requirements:**

* Expand game availability from Android to web-based platforms.
* Allow multiplayer games involving multiple teams and players.
* Preserve a consistent gameplay structure with four timed rounds.

**Technical Requirements:**

* Ensure only one instance of the game is running at a time.
* Enforce unique names for games, teams, and players.
* Support real-time interactions across multiple platforms.
* Provide scalable, secure communication and data handling.

## [Design Constraints](#_xnireub9qde3)

**Single Game Instance:** Only one instance of a game should exist at a time. This is achieved using the Singleton design pattern, which limits the instantiation of the Game class to one object and provides a global access point.

**Name Uniqueness:** Team and player names must be unique. This requires centralized name validation and assignment using UUIDs and/or a name registry to ensure no conflicts in distributed environments.

**Web-Based Distribution:** The application must be accessible via web browsers across various devices. This limits the design to platform-independent languages like JavaScript, HTML5, and CSS, and requires backend logic compatible with platforms like Node.js or Python (Flask/Django).

**Concurrency:** As the game supports multiple players in real time, the system must handle concurrent user interactions and synchronize game state across all clients using technologies such as WebSockets.

**Security:** The system must secure user and gameplay data through encryption, secure login protocols, and validation mechanisms across network communication.

## [System Architecture View](#_9wq5fhi9gjpf)

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](#_w0k1wehzwtrb)

The UML diagram provided introduces four main classes: Entity, Game, Team, and Player.

* **Entity** is a base class containing shared attributes such as id and name. This supports the principle of inheritance.
* **Game** inherits from Entity and serves as the central hub, containing a list of Teams. It implements the Singleton pattern to enforce a single active instance.
* **Team** also inherits from Entity and manages a collection of Player instances. It supports multiple players per team.
* **Player**, inheriting from Entity, represents an individual user assigned to a specific team.

**OOP Principles Used:**

* **Inheritance** (via the Entity superclass) avoids redundancy.
* **Encapsulation** keeps data and methods bundled in each class.
* **Abstraction** simplifies complexity by focusing on essential behaviors.
* **Design Patterns** such as Singleton enforce architectural constraints efficiently.

**"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](#_38i95qi79o1s)

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 can support web servers like Apache or Nginx and development tools like Node.js and Docker. However, it is less common for production deployment due to higher costs and reliance on Apple hardware. It's generally better suited for development environments than for hosting scalable web services. | Linux is the most preferred OS for production web hosting. It supports robust and scalable technologies like Apache, Nginx, and Docker. It's open-source and used widely in cloud environments like AWS, Azure, and GCP. | Windows Server supports web applications via IIS and .NET. While licensing costs apply, it integrates well with enterprise systems. Not as lightweight as Linux but can be a viable option for Microsoft tech stacks. | Mobile devices are clients, not typically used for server hosting. Instead, they connect to cloud-based or hosted web servers. RESTful APIs and WebSocket connections facilitate real-time communication with the backend. |
| **Client Side** | Supports modern web browsers like Safari, Chrome, and Firefox. Some compatibility testing is needed for Safari-specific behavior. macOS users expect high-quality UX, so responsive design and browser support are crucial. | Fully supports modern web browsers like Chrome and Firefox. Offers a reliable and secure environment for browser-based gameplay. Responsive design ensures compatibility across distributions. | Supports all major web browsers, including Chrome, Edge, and Firefox. Popular among gamers and casual desktop users. Responsive HTML/CSS ensures consistent UI on different screen resolutions. | A responsive HTML5 interface ensures compatibility with mobile browsers on Android and iOS. Touch input, screen responsiveness, and performance optimization are critical for user experience. |
| **Development Tools** | Tools like Xcode (for iOS), Visual Studio Code, and Docker are widely used. While Xcode is Mac-specific, cross-platform tools like React Native can also be developed on macOS. No major licensing costs for basic usage. | Linux supports open-source IDEs and programming environments including VS Code, Eclipse, and JetBrains. Languages like JavaScript, Python, and Java are common. Free and community-supported, reducing overhead. | Visual Studio, .NET, and Docker are widely used. Visual Studio Community is free for small teams. Strong support for enterprise-level development and integration with Azure services. | Cross-platform frameworks like Flutter and React Native allow deployment on both iOS and Android. Native development uses Android Studio (Java/Kotlin) and Xcode (Swift). Xcode requires macOS. |

## 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**: Linux is recommended for server deployment. It offers excellent stability, scalability, and compatibility with containerized tools like Docker and Kubernetes.
2. **Operating Systems Architectures**: Linux follows a modular architecture with isolated user and kernel space. Its architecture supports multithreading and asynchronous communication, ideal for a distributed web game.
3. **Storage Management**:A relational database like PostgreSQL or MySQL is recommended for storing persistent data (users, teams, scores). These databases support ACID properties and efficient querying for unique name validation.
4. **Memory Management**: Linux uses efficient memory techniques including virtual memory, caching, and memory paging. This ensures smooth performance, especially under load with many concurrent users.
5. **Distributed Systems and Networks**: WebSockets or SignalR will facilitate real-time updates across devices. The application will run as microservices or modules to ensure scalability and fault tolerance, supported by load balancers and container orchestration.
6. **Security**: Security protocols will include HTTPS for encrypted communication, token-based authentication, input validation, and role-based access control. Sensitive user data will be encrypted in storage and during transmission.

Operating Platform:   
Linux is the recommended server operating platform for The Gaming Room. It is widely adopted for enterprise-level hosting due to its high scalability, strong performance, and low total cost of ownership. As an open-source OS, Linux provides flexibility, robust community support, and compatibility with containerized technologies like Docker and orchestration tools like Kubernetes, which are essential for distributed web applications.  
  
Operating Systems Architectures:   
Linux follows a modular monolithic kernel architecture, which separates user and kernel space and supports modular loading of drivers and services. This architecture allows efficient scheduling, multiprocessing, and low-level control over system resources. Its support for multithreading, asynchronous I/O, and background daemon processes makes it ideal for real-time multiplayer web games like Draw It or Lose It.  
  
Storage Management:   
A relational database management system (RDBMS) such as PostgreSQL or MySQL is recommended. Both are ACID-compliant, ensuring data integrity, especially when multiple users interact concurrently. These systems support indexing, schema constraints, and structured queries for efficient data retrieval. Linux also offers advanced file systems (e.g., ext4, XFS) and tools like Logical Volume Manager (LVM) for scalable and secure storage management.  
  
Memory Management:   
Linux utilizes a sophisticated memory management system including paging, virtual memory, and dynamic allocation. The kernel dynamically assigns memory to processes based on need, using caches and swap space to maintain system responsiveness. These features are critical to maintaining server performance as the number of simultaneous game sessions grows.  
  
Distributed Systems and Networks:   
The application will run as a distributed system using microservices. Each service—such as user management, game state, or scoring—can be containerized and deployed independently. Services will communicate over internal RESTful APIs or WebSockets for low-latency updates. Kubernetes can manage scalability and availability, while load balancers distribute traffic across nodes. Network fault tolerance is addressed using redundancy and failover configurations.  
  
Security:   
Data security will be maintained with HTTPS/TLS encryption in transit and AES-256 encryption at rest. The system will use token-based authentication (e.g., JWT) for secure session management and enforce role-based access control to prevent unauthorized actions. Linux offers additional hardening tools like SELinux, iptables, and auditd for logging and access monitoring. Regular code audits, vulnerability scanning, and input sanitization will be part of the secure development lifecycle.