

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 | 06/28/2025 | Jack Becker | Initial version of software design document for Draw It or Lose It project. |
| 2.0 | 06/08/2025 | Jack Becker | Updated software design document for Project Two: Enhanced Evaluation and Recommendations. |
| 3.0 | 06/22/2025 | Jack Becker | Updated software design document for Project Three: Enhanced 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 a web-based version of their popular Android app Draw It or Lose It. The application must support multiple games, each containing multiple teams and players. To ensure a seamless transition and maintain robust performance across platforms, an object-oriented Java application has been developed with a modular and scalable structure.

This design uses core OOP principles—abstraction, inheritance, encapsulation, and the Singleton pattern—to manage game logic. The solution ensures name uniqueness, game integrity, and scalable multi-user functionality. This system prepares the game for deployment across various operating systems and web browsers, making it accessible on desktop and mobile environments.

## Requirements

* *A player will not be able to join more than one team.*
* *Games and teams must maintain unique names.*
* *Only one instance of the game service should exist at any time.*
* *The solution must support multiple games and players concurrently.*
* *Teams can contain multiple players.*
* *The system must allow team name checks and prevent duplication.*
* *The application must run on Windows, Linux, macOS, and mobile devices.*

## [Design Constraints](#_2et92p0)

* Platform Independence: The solution must run on various operating systems. This affects the technology stack, requiring it to be based on Java or another cross-platform language.
* Name Uniqueness: Must enforce unique names for games, teams, and players to prevent logical conflicts.
* Centralized Game Management: Only one GameService instance must control all game data. This is handled via the Singleton pattern.
* Scalability: The solution must accommodate additional players or teams without redesign.
* Data Integrity: Must preserve relationships between games, teams, and players without overlap or loss.

## [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)

* The UML class diagram includes:
* Entity (abstract): Contains id and name.
* Game: Extends Entity. Holds a list of teams.
* Team: Extends Entity. Holds a list of players.
* Player: Extends Entity.
* GameService: Singleton class responsible for managing games, teams, and players.
* ProgramDriver: Entry point to simulate game setup and testing.

OOP Principles:

Inheritance: Entity is extended by Game, Team, and Player to avoid duplication.

Encapsulation: Each class protects and controls its data.

Abstraction: Users interact with high-level methods rather than implementation details.

Singleton Pattern: Ensures a single instance of GameService to centralize control.

**"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) (Updated)

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 supports Apache and NGINX for web-based deployment, but is not a common production choice. Server licensing costs are high due to Apple hardware requirements. | Linux is the ideal platform for server deployment—robust, free to use (open-source), and highly scalable with strong support for Apache, NGINX, and Tomcat. No licensing costs. | Windows Server supports IIS, Tomcat, and other web servers. Licensing costs are significant (Windows Server licenses and CALs required). Heavier overhead than Linux. | Mobile devices are not used as servers. They are clients only; limited server deployment options. |
| **Client Side** | Mac client app must be built as a responsive web app for Safari and desktop Chrome/Firefox. Cost is moderate—need Mac-specific testing. Expertise in cross-browser testing required. | Linux clients access the app through Firefox or Chrome. Cost-effective and easy to support. Minimal extra expertise required; good compatibility. | Windows clients access the app through Edge, Chrome, Firefox. High compatibility; minimal extra expertise required. Testing needed for Edge quirks. | Mobile apps must be responsive, lightweight HTML5 apps or native apps. Requires additional development effort: Android (Java/Kotlin), iOS (Swift). Expertise in mobile app frameworks (React Native, Flutter) is helpful. |
| **Development Tools** | IntelliJ IDEA, Eclipse, Apache Tomcat. Java is primary language. No significant licensing costs for Java stack. | Eclipse, IntelliJ, Maven, Git. No licensing costs; open-source tools favored. Single dev team can manage both server and Linux client needs. | Visual Studio Code, Eclipse, Java, Tomcat. No licensing costs. Windows and Linux teams can share tools easily. | Android Studio (Java/Kotlin), Xcode (Swift). These tools are platform-specific—requires distinct mobile development team. Apple developer license ($99/year) required for iOS deployment. Android deployment free. |

## Recommendations (Updated)

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 server-side deployment of Draw It or Lose It, Linux is the recommended operating platform due to its unmatched stability, security, and cost-efficiency. Enterprise-grade distributions such as Ubuntu Server LTS or Red Hat Enterprise Linux are widely adopted in industry and offer strong compatibility with essential tools like Docker, Kubernetes, and CI/CD platforms. These distributions also have excellent long-term support and integration with cloud providers such as AWS, Microsoft Azure, and Google Cloud Platform. Unlike proprietary operating systems, Linux is open-source, eliminating licensing costs while providing extensive community and commercial support. Its performance under heavy concurrent loads and ability to handle large-scale distributed applications make it an ideal choice for hosting the game’s backend services in both development and production environments.
2. **Operating Systems Architectures**: The most effective system architecture for Draw It or Lose It is a distributed, service-oriented architecture (SOA) designed for cross-platform access. The backend services will be exposed through stateless RESTful APIs built in Java using Spring Boot, while the frontend will consist of a responsive web application developed with React.js. For mobile users, a cross-platform development framework such as Flutter is recommended to deliver high-performance native-like experiences with a single codebase. This separation of concerns between the frontend and backend promotes modularity and allows for independent updates without disrupting system-wide functionality. The use of containerization and microservices supports rapid deployment and scaling, and the system will leverage HTTPS for secure communication between all components. Together, this architecture ensures maximum maintainability, device compatibility, and scalability.
3. **Storage Management**: The application should implement a dual-layered storage model, combining relational and object-based storage solutions to optimize performance and data organization. A cloud-hosted relational database management system (RDBMS), such as PostgreSQL or MySQL, should be used for structured data including game sessions, users, teams, and scoring information. These databases support ACID-compliant transactions and offer strong indexing, replication, and backup capabilities. For unstructured and media-rich content like user-generated drawings or assets, a cloud object storage solution like Amazon S3 is recommended due to its global availability, high durability, and low latency. This hybrid strategy ensures fast, reliable access to dynamic data while efficiently managing large static content. Performance tuning through query optimization, caching (e.g., Redis), and content delivery via CDN will ensure the game remains responsive under peak usage.
4. **Memory Management:** Efficient memory management is critical for supporting real-time, multiplayer gameplay at scale. Linux provides powerful memory management features, such as virtual memory paging, shared memory control, and memory cgroups that ensure optimal use of system resources. On the application side, the Java Virtual Machine (JVM) will be tuned with custom garbage collection settings and heap size configurations based on performance profiling and usage patterns. Within the game’s backend code, memory-efficient data structures and lifecycle-aware programming practices—such as lazy initialization, weak references, and object pooling—will help minimize unnecessary allocations. Continuous memory monitoring using tools like Prometheus and Grafana will allow the development team to detect memory leaks, garbage collection pauses, and performance regressions in real time, enabling proactive maintenance and ensuring seamless user experiences.
5. **Distributed Systems and Networks**: To support a global user base and maintain high availability, Draw It or Lose It should adopt a distributed system architecture deployed via container orchestration platforms like Kubernetes. Backend services will be distributed across multiple instances and geographic regions, allowing for redundancy and failover capabilities. Secure, stateless communication over HTTPS will be maintained between client applications and backend APIs, using JSON Web Tokens (JWT) for session management. Load balancing will be handled by reverse proxies such as NGINX or HAProxy, ensuring consistent request routing and scaling based on user demand. The use of service meshes like Istio can further enhance service discovery, monitoring, and traffic shaping. To reduce latency and improve load times, a Content Delivery Network (CDN) such as Cloudflare or AWS CloudFront should be used to serve static assets from edge locations, bringing content physically closer to users worldwide. These technologies collectively create a resilient, scalable, and efficient distributed environment for the game.
6. **Security**: Security must be integrated at all layers of the system to protect both users and data from evolving threats. All communication between clients and servers will be encrypted using TLS 1.3 to ensure confidentiality and integrity. Sensitive data stored on servers, including user information and game state data, will be encrypted at rest using AES-256. User authentication will be handled using secure, modern practices including OAuth 2.0 for session authorization and Argon2 for password hashing, along with salted storage to protect against brute force attacks. At the infrastructure level, Linux hosts will be hardened using SELinux or AppArmor, and network traffic will be controlled using iptables or nftables firewalls. Security logging and auditing will be centralized using tools like the ELK Stack or Splunk, enabling real-time alerting and forensic analysis. Input validation and output encoding will prevent injection and XSS attacks, while a strict role-based access control (RBAC) policy will limit sensitive operations to authenticated and authorized users. Regular penetration tests, vulnerability scans, and dependency audits will be conducted as part of the development cycle to ensure the system adheres to best practices in cybersecurity.
7. **Development Process:** To ensure maintainability, scalability, and continuous improvement, Draw It or Lose It should adopt a modular, agile-based development process structured around continuous integration and continuous deployment (CI/CD) pipelines. The core backend—responsible for managing user sessions, game logic, and data persistence—should be developed using Java with Spring Boot, taking advantage of its strong ecosystem, REST API support, and compatibility with microservice deployment patterns. The frontend web application should be developed with React.js to allow a component-based architecture and rapid user interface prototyping, while mobile applications should be built with Flutter to enable shared codebases and faster deployment cycles across iOS and Android. Development teams should be organized into cross-functional units focusing on backend, frontend, and mobile development. This separation of responsibilities fosters domain-specific expertise while ensuring alignment through shared API specifications and version-controlled contracts. Version control should be managed via Git, with platforms like GitHub or GitLab supporting automated CI/CD workflows. Automated testing frameworks such as JUnit (for backend), Jest (for frontend), and integration testing suites should be incorporated into the pipeline to guarantee stability and code quality at every commit. Each module should undergo code reviews, unit testing, and containerized deployment in test environments before being promoted to production. Deployment to production should leverage infrastructure-as-code tools like Terraform or AWS CloudFormation to ensure consistent environments across staging and live systems. Agile methodologies like Scrum or Kanban will help drive incremental development and allow the team to respond quickly to changing requirements or user feedback. Regular sprint reviews, retrospectives, and backlog grooming sessions will support transparency and continual process improvement. By integrating these tools and practices, the development process will remain flexible, transparent, and robust enough to support both current goals and future scaling needs.