

***“Draw It or Lose It”***

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 11/17/2024 | Michael Cary | Initial draft completed, including the Executive Summary, Requirements, Design Constraints, Domain Model analysis, Evaluation, and Recommendations sections. Content structured to address client requirements and align with the project guidelines. |
| 1.1 | 11/29/2024 | Michael Cary | Updated Evaluation section with refined platform analysis for server and client considerations. Updated Recommendations to align with new findings while maintaining original structure. Adjusted Executive Summary, Requirements, and Design Constraints for consistency with cross-platform deployment strategy. |
| 1.2 | 12/14/2024 | Michael Cary | Enhanced Recommendations section and ensured that it adequately addresses Project 3 guidelines, focusing on system architectures, storage management, memory management, distributed systems, and security considerations for "Draw It or Lose It." Also ensured the document was free of errors. |

**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 is transitioning its popular game, *Draw It of Lose It*, from an Android only based application to a web-based platform. This will allow it to be accessible on multiple devices. The challenge lies in ensuring scalability, unique naming conventions for games, teams, and players and maintaining a single instance of the game in memory. To address these needs, I propose a responsive web design leveraging modern frameworks like React. On the server side, Linux is recommended for its cost-efficiency, scalability, and stability. This approach ensures seamless integration into a distributed environment while adhering to the client’s requirements for unique identifiers, robust memory management, and multiuser functionality.

## Requirements

1. The game must support multiple teams with multiple players per team.
2. Game, team, and player names must remain unique.
3. The solution should operate as a single-instance application in memory.
4. The application must be compatible with Linux, Mac, Windows, and mobile platforms using responsive web technologies.
5. The back-end must scale to support thousands of concurrent users in a distributed environment.

## [Design Constraints](#_2et92p0)

1. **Distributed Environment:**

The application must function in a web-based environment that is multiuser, requiring RESTful APIs for seamless communication between client and server.

1. **Scalability:**

Efficient back-end handling for thousands of concurrent users is critical. Linux servers offer the best performance-to-cost ratio.

1. **Cross-Platform Compatibility:**

The client-facing application must utilize responsive web technologies to ensure compatibility across desktop browsers (Linux, Mac, Windows) and mobile devices (iOS, Android).

1. **Development Tools:**

Java is the primary language for back-end development, and React Native ensures consistency for mobile apps.

## [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 illustrates the structure and relationships of the game application’s components. At its core, the Entity class serves as a base class, encapsulating common attributes such as id and name to ensure reusability and consistency across the derived classes: Game, Team, and Player. The “GameService” class utilizes the Singleton Design Pattern, ensuring that only one instance of the service exists in memory at any time, which is critical for efficient resource management in a distributed environment. This class is responsible for managing collections of games, teams, and players, enabling the creation of unique identifiers and ensuring that names are distinct across entities. The design demonstrates fundamental object-oriented principles. Inheritance reduces redundancy by allowing Game, Team, and Player to inherit shared properties from Entity. Encapsulation secures data with private fields accessed through getters. Polymorphism allows customization of behavior through overridden “toString()” methods. Additionally, the use of the Iterator Pattern facilitates the traversal and validation of entities to enforce uniqueness and streamline entity management. This design ensures scalability, maintainability, and alignment with the client’s requirements.

**"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** | Reliable and secure, with a strong ecosystem for development. Higher costs for hosting and support. | Ideal for server environments; cost-effective, highly scalable, and widely supported for web hosting. | Easy to set up with built-in tools; widely used in businesses. Stability concerns under high scalability requirements. | Limited to client-side functionality; third-party hosting required for server features. |
| **Client Side** | Moderate cost; development expertise in macOS is needed. Excellent browser compatibility. | Cost-efficient, open-source, and reliable for end-users. Development expertise required for compatibility. | User-friendly for developers, common in enterprises. Broad compatibility but potential browser inconsistencies. | Crucial platform for the target audience. Responsive design frameworks ensure compatibility across iOS and Android. |
| **Development Tools** | Java (cross-platform), Xcode for iOS, and IDEs like IntelliJ. | Java (cross-platform), Eclipse, and open-source debugging tools. | Java, Eclipse, Visual Studio (user-friendly for Windows-based systems). | Requires frameworks for responsive design (e.g., Bootstrap, React Native). Strong focus on device testing tools. |

**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**:

Use Linux for server-side deployment due to its superior cost-efficiency, reliability, and scalability. Its strong support for web hosting and open-source tools makes it the ideal choice for running a distributed, scalable system.

1. **Operating Systems Architectures**:

Adopt a multi-tier architecture separating the client, server, and database layers. This ensures modularity, allowing the application to scale effectively and support thousands of concurrent users. RESTful APIs will handle communication between these tiers.

1. **Storage Management**:

Implement a relational database such as MySQL to efficiently manage unique names for games, teams, and players. This database can scale horizontally to handle increasing data volumes and ensure consistency across distributed systems.

1. **Memory Management**:

Utilize the Singleton Design Pattern to ensure only one instance of the game service exists in memory at any given time. Combine this with caching strategies to optimize resource use and reduce server load.

1. **Distributed Systems and Networks**:

Leverage REST APIs to enable communication between distributed components. This approach ensures compatibility across Linux, Mac, Windows, and mobile platforms while maintaining performance and scalability in a web-based environment.

1. **Security**:

Protect sensitive user data by encrypting credentials and implementing authentication mechanisms such as OAuth2 tokens. Use HTTPS to secure data in transit and apply robust input validation to prevent common security vulnerabilities.