

# 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 | 12/13/2024 | Zeb Hawthorne | Project Three Final Draft |

**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 expanding its interactive game, "Draw It or Lose It," to support a web-based distributed environment. The current software operates on a single platform, limiting scalability and access. The goal is to design and develop a robust, platform-independent solution that meets business requirements, enhances the user experience, and ensures secure, scalable, and efficient performance across multiple devices. This document outlines the design problem and presents a viable solution to implement a scalable and secure application using object-oriented principles, distributed architecture, and role-based access.

## Requirements

* Expand the game to a web-based distributed platform.
* Ensure scalability to handle increasing users and game sessions.
* Provide secure access to users’ data and game functionalities.
* Develop the software using object-oriented programming.
* Ensure compatibility with multiple operating systems (Windows, Linux, macOS, and mobile devices).
* Use a secure, distributed network architecture.
* Optimize memory and storage management.

## [Design Constraints](#_2et92p0)

Developing "Draw It or Lose It" for a web-based distributed environment presents several design constraints. The software must support Windows, Linux, macOS, and mobile devices, requiring the use of cross-platform development tools and languages like Java. Efficient memory and storage management are necessary to handle simultaneous game sessions without performance degradation. Security is a critical consideration, with user authentication, data encryption, and secure communication protocols required to protect sensitive data. Ensuring seamless communication between distributed components with minimal latency is essential, while the system must also accommodate future growth in users and game complexity through a modular and scalable architecture. These constraints influence the choice of programming languages, frameworks, and deployment strategies to balance functionality, performance, and cost.

## [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)

1. Entity Class:
   * Attributes: Contains common attributes (id, name) shared by other classes.
   * Methods: Provides basic operations (getId, getName, toString).
   * Purpose: Promotes code reusability by serving as a base class for Game, Team, and Player.
2. GameService Class:
   * Attributes: Manages lists of games and tracks unique identifiers for Game, Team, and Player.
   * Methods: Provides operations to add and retrieve games and teams (addGame, getGame, etc…). Implements the Singleton design pattern to ensure only one instance of GameService exists.
3. Game Class:
   * Attributes: Maintains a list of teams participating in a game.
   * Methods: Includes methods to add teams and represent the game as a string.
   * Relationship: Extends the Entity class, inheriting common attributes and behaviors.
4. Team Class:
   * Attributes: Holds a list of players.
   * Methods: Allows adding players and retrieving team details.
   * Relationship: Extends Entity, promoting reusability and consistency.
5. Player Class:
   * Attributes: Stores unique player identifiers and names.
   * Methods: Provides string representation of player details.
   * Relationship: Represents the lowest level of the hierarchy with no extensions.
6. ProgramDriver and SingletonTester Classes:
   * Provide the entry point for testing and initializing the application using Singleton design pattern for controlled access.

Object-Oriented Principles Demonstrated:

* Encapsulation: Each class defines specific attributes and methods, ensuring data integrity and controlled access.
* Inheritance: The Entity class reduces redundancy by providing common attributes and methods to its subclasses.
* Polymorphism: Subclasses (Game, Team, Player) override or extend base class methods as needed.
* Abstraction: Higher-level operations are abstracted within the GameService class, hiding implementation details from the user.

**"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** | User-friendly interface, strong performance; limited native support for some frameworks; high hardware cost. | Highly customizable, excellent performance and security; requires expertise; broad framework support. | Widely used, strong compatibility with tools and frameworks; moderate security risks; higher cost than Linux. | Enables portability and broad accessibility; limited resources for complex computations; platform-specific development needs. |
| **Client Side** | Higher cost; moderate time for development; requires expertise in macOS-specific tools like Xcode. | Low cost; moderate time; requires knowledge of open-source tools and environments. | Moderate cost and time; extensive tools available; requires familiarity with .NET or similar frameworks. | Higher cost for cross-platform support; longer development time; requires expertise in frameworks like Flutter or React Native. |
| **Development Tools** | Java, Xcode, IntelliJ IDEA | Java, Eclipse, IntelliJ IDEA | Java, Visual Studio, IntelliJ IDEA | Java, Kotlin, Swift, 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:

1. **Operating Platform**: Linux is recommended for hosting the distributed application due to its performance, cost efficiency, and security.
2. **Operating Systems Architectures**: Linux’s modular architecture supports scalability and distributed systems.
3. **Storage Management**: Implement cloud-based storage solutions like AWS S3 for scalable, secure data management.
4. **Memory Management**: Use Java’s Garbage Collection and efficient memory allocation techniques to optimize performance.
5. **Distributed Systems and Networks**: Employ RESTful APIs and WebSocket protocols to enable seamless communication between distributed components.
6. **Security**: Use HTTPS, OAuth 2.0 for authentication, and encrypt sensitive data to protect users across platforms.