

The Gaming Room

# **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 | 03/22/2025 | Ann Jessica Tan | Initial draft of the software design document |
| 1.1 | 04/05/2025 | Ann Jessica Tan | Added platform evaluation, development tools analysis and recommendations |
| 1.2 | 0420/2025 | Ann Jessica Tan | Completed system architecture recommendations, including OS architecture, memory/storage management, distributed systems, and security considerations |

## 

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has approached Creative Technology Solutions (CTS) to develop a web-based version of their popular game, *Draw It or Lose It*, which is currently available only as an Android app. The game involves teams competing to guess phrases, titles, or things based on rendered images. The client has specific requirements, including the ability to support multiple teams and players, ensure unique names for games and teams and maintain only one instance of the game in memory at any given time.

Our team has analyzed the requirements and proposes a solution that leverages object-oriented programming principles and design patterns to create a scalable, efficient and maintainable application. This document outlines the design constraints, system architecture, and domain model to guide the development process.

## Requirements

*The client has outlined the following* ***business and technical requirements****:*

1. ***Business Requirements****:*
   * *Develop a web-based version of Draw It or Lose It that supports multiple platforms.*
   * *Ensure the game can handle multiple teams and players.*
   * *Provide a unique naming system for games and teams to avoid conflicts.*
2. ***Technical Requirements****:*
   * *Only one instance of the game can exist in memory at any given time.*
   * *Use unique identifiers for games, teams, and players.*
   * *Ensure the application is scalable and can be deployed in a distributed environment.*

## [Design Constraints](#_2et92p0)

The following design constraints have been identified for developing the game application in a web-based distributed environment:

1. **Singleton Pattern**:
   * The GameService class must ensure that only one instance of the game exists in memory. This is achieved using the Singleton design pattern, which restricts instantiation of the class to a single object.
2. **Unique Identifiers**:
   * Each game, team, and player must have a unique identifier to prevent conflicts and ensure proper management of resources.
3. **Platform Compatibility**:
   * The application must be compatible with multiple platforms, including web browsers and mobile devices. This requires careful consideration of cross-platform development tools and frameworks.
4. **Scalability**:
   * The application must be designed to handle a growing number of users and games. This involves optimizing database interactions and ensuring efficient memory management.
5. **Security**:
   * User information must be protected, and the application must implement security measures to prevent unauthorized access.

## [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 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.**

1. **Entity Class:**
   * The Entity class is a base class that holds common attributes (id and name) and behaviors. It serves as the parent class for Game, Team, and Player.
2. **GameService Class:**
   * The GameService class manages the list of active games and ensures only one instance exists using the Singleton pattern. It provides methods to add games, retrieve games by ID or name, and manage unique identifiers for games, teams, and players.
3. **Game Class:**
   * The Game class represents a game instance and contains a list of teams. It inherits from the Entity class and provides methods to add teams and retrieve game details.
4. **Team Class:**
   * The Team class represents a team within a game and contains a list of players. It inherits from the Entity class and provides methods to add players and retrieve team details.
5. **Player Class:**
   * The Player class represents a player within a team. It inherits from the Entity class and provides methods to retrieve player details.
6. **ProgramDriver and SingletonTester Classes:**
   * The ProgramDriver class is the entry point of the application and uses the SingletonTester class to verify the Singleton implementation.

**Object-Oriented Principles Demonstrated:**

* **Inheritance:** The Game, Team, and Player classes inherit from the Entity class, promoting code reuse and reducing redundancy.
* **Encapsulation:** Attributes like id and name are encapsulated within the Entity class, ensuring controlled access through getter methods.
* **Singleton Pattern:** The GameService class uses the Singleton pattern to ensure only one instance exists in memory, fulfilling the requirement of a single game instance.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is known for its stability and performance, making it suitable for hosting web-based applications. However, it can be costly.  **UPDATE**:  macOS Server is stable but limited in scalability and incurs high hardware/licensing costs (e.g., Apple hardware requirements). Not recommended for large-scale deployments. | Linux is highly customizable and cost-effective, making it a popular choice for server-side development.  **UPDATE**:  Linux is ideal for server hosting due to its open-source nature, cost-effectiveness, and scalability. It supports robust web servers like Apache and Nginx, making it a top choice for deploying the game’s backend. No licensing fees reduce overhead for The Gaming Room. | Windows offers robust support for enterprise applications but can be expensive.  **UPDATE:**  Windows Server offers strong enterprise support and integrates well with .NET frameworks but licensing costs (e.g., Windows Server licenses, SQL Server) could strain the budget. Suitable if the team already uses Microsoft tools. | Mobile devices are not typically used for server-side hosting but are essential for client-side access.  **UPDATE:**  Mobile devices (Android/iOS) are not viable for server hosting but will serve as critical client endpoints. |
| **Client Side** | Mac clients require development tools like Xcode, which can increase development time and cost.  **UPDATE**:  Mac clients work seamlessly with Safari and other browsers but require testing for macOS-specific quirks. Development costs are higher due to Xcode dependency. | Linux clients are cost-effective but may require additional expertise.  **UPDATE**:  Linux desktop users can access the game via web browsers (Chrome, Firefox). However, fragmentation across distributions may require additional testing. | Windows clients are widely used and supported, reducing development complexity.  **UPDATE**:  Windows has broad browser support (Edge, Chrome, Firefox) and is the most widely used OS for desktops, ensuring easy adoption. | Mobile devices require cross-platform development tools like React Native or Flutter.  **UPDATE**:  Android and iOS require a responsive HTML5 interface or cross-platform frameworks (e.g., React Native) to ensure consistent performance across devices. |
| **Development Tools** | Xcode is the primary IDE for Mac development, supporting languages like Swift and Objective-C.  **UPDATE**:  Xcode (free but macOS-exclusive) is required for iOS testing, adding complexity for non-Mac developers. | Linux supports a wide range of IDEs and tools, including Eclipse and IntelliJ IDEA.  **UPDATE**:  Free, open-source tools (Eclipse, IntelliJ IDEA) reduce costs. Supports Java, Python, and Node.js, but may require expertise in Linux environments. | Windows supports Visual Studio, a powerful IDE for .NET and C# development.  **UPDATE**:  Visual Studio (licensed) simplifies .NET/C# development. Broad IDE support but potential licensing fees for proprietary tools. | Mobile development tools include Android Studio for Android and Xcode for iOS.  **UPDATE**:  Android Studio (free) for Android, Xcode for iOS. Cross-platform tools (Flutter, React Native) can streamline development but may limit native features. |

**Recommendations**

As part of our ongoing collaboration with The Gaming Room, our team at Creative Technology Solutions has thoroughly evaluated the technical requirements and constraints for expanding *Draw It or Lose It* across multiple platforms. Based on our analysis, we are recommending the following solutions to ensure scalability, performance, and security in a distributed web-based environment.

1. **Operating Platform**

We recommend Linux as the operating platform for server-side hosting. Linux offers high scalability, stability, and performance, which are critical as the game grows and reaches more users. It is also open-source, which significantly reduces licensing costs and overhead, making it a budget-friendly option. Linux supports a wide range of modern web servers (such as Nginx and Apache) and works seamlessly with cloud providers like AWS, Google Cloud, and Azure.

1. **Operating Systems Architectures**

Linux supports various system architectures, including x86\_64 and ARM, which makes it highly adaptable to different hardware environments. Its modular architecture allows system components such as kernel modules and device drivers, to be added or removed as needed, which enhances performance and flexibility. For Draw It or Lose It, we recommend using a microservices-based architecture. This allows different parts of the application (e.g., game logic, user authentication, team management) to run independently, enabling smoother deployment, updates and scaling across different operating environments.

1. **Storage Management**

To efficiently manage player data, game sessions, teams, and unique identifiers, we recommend using a Relational Database Management System (RDBMS) such as PostgreSQL or MySQL. These databases support structured queries and offer robust features like data integrity, transaction control and scalability. Combined with regular backups and database replication for fault tolerance, this setup ensures reliable storage management aligned with the application’s performance needs.

1. **Memory Management**

Linux offers advanced memory management techniques that align well with the constraints of Draw It or Lose It, specifically the requirement for only one instance of the game to be in memory. It uses virtual memory, paging and swap space to maximize performance and efficiency. The Singleton design pattern already implemented in the GameService class complements these techniques by ensuring the application maintains a minimal memory footprint while still being responsive.

1. **Distributed Systems and Networks**

To support communication between different client platforms (web, Android, iOS), we recommend implementing a distributed architecture using RESTful APIs. REST APIs enable platform-agnostic communication over HTTP, allowing seamless data exchange between the server and different client devices.

Additionally, we suggest using cloud-based services (such as AWS Elastic Load Balancing and Auto Scaling) to ensure high availability and fault tolerance. These services help the game remain accessible even during high traffic or server outages. Using message queues (e.g., RabbitMQ or Apache Kafka) can also ensure that messages between services are not lost during temporary connectivity issues.

1. **Security**

Security is a top priority, especially since the game will store and transmit user data across networks. We recommend the following measures:

* SSL/TLS encryption for all communication between clients and servers to protect data in transit.
* OAuth 2.0 for secure authentication and authorization, allowing users to log in via trusted providers (e.g., Google, Facebook) without exposing their credentials.
* Input validation and sanitization to prevent common web vulnerabilities such as SQL injection and cross-site scripting (XSS).
* Role-based access control (RBAC) to restrict access to sensitive data and operations.
* Regular security audits and patch management to address known vulnerabilities in third-party libraries and operating system components.

By following these recommendations, Draw It or Lose It will be well-positioned for a secure, scalable, and reliable multi-platform deployment.