

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 | 1/22/2025 | Shamus Cerny | Initial Creation of Design Document |
| 2.0 | 2/3/2025 | Shamus Cerny | Feedback Implementation / Project Two |
| 3.0 | 2/20/2025 | Shamus Cerny | Project Three / 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)

## Gaming Room wants to develop a web-based version of *Draw It or Lose It*, their existing Android-only multiplayer game inspired by *Win, Lose or Draw*. In the game, multiple teams compete to guess a phrase, title, or concept based on a drawing, with rounds lasting one minute and opportunities for other teams to guess if the primary team fails. The client requires assistance in setting up the development environment, including both software and hardware infrastructure, to support smooth gameplay. The new version will feature a large library of stock drawings that must render at a steady rate. This document outlines the system architecture, hardware considerations, and recommendations to ensure a seamless and efficient implementation.

## Requirements

The client's technical requirements for the game application are as follows:

1. **One or More Teams per Game**: A game will support multiple teams, each competing to guess the drawing. Each team is formed dynamically and can contain multiple players.
2. **Unique Game and Team Names**: The game and team names must be unique across all active games. This allows the application to verify that names do not conflict when a user tries to create a new team or game.
3. **Unique Identifiers for Each Entity**: Every game, team, and player needs a unique identifier. This ensures that only one instance of a game can exist in memory, and no conflicts occur with player or team identity.
4. **Web-Based Platform**: The game application needs to be developed in a web-based environment to support multiple devices and platforms.

## [Design Constraints](#_2et92p0)

* 1. **Web-Based Deployment:** The application must be accessible from various devices, including desktops and mobile phones, requiring compatibility across multiple browsers and stable client-server communication.
  2. **Infrastructure and Scalability:** Transitioning from a mobile-only architecture to a web-based system requires a scalable infrastructure, either on-premises or cloud-based, with dedicated environments for development, testing, and production. The system must efficiently handle increased storage and memory demands to support rendering a large library of stock images at a performance level comparable to or exceeding the current mobile version.
  3. **Security and Authentication:** Unlike the Android version, which relies on device-based authentication, the web-based application will require a robust authentication system for multiple players and teams. Security measures must include secure user authentication, data protection, and restrictions to prevent multiple logins from different devices.
  4. **Administrative Complexity:** The web-based version introduces new administrative requirements, including an interface for managing users, teams, and game sessions, ensuring proper access control and user management.

## [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 for the Draw It or Lose It application demonstrates the relationships between several key classes. **ProgramDriver** uses **SingletonTester** to test the singleton behavior of **GameService**, ensuring only one instance exists. **Game**, **Team**, and **Player** inherit from the base class **Entity**, which provides shared attributes like id and name. **GameService** manages multiple **Game** objects, each containing multiple **Team** objects, which in turn contain **Player** objects. This uses **aggregation** to represent the "has-a" relationship, while **inheritance** ensures code reuse for common attributes. The **singleton pattern** in **GameService** ensures a single instance is maintained, fulfilling the requirement for unique identifiers and controlled memory usage. **Encapsulation** protects internal state, and **dependency** between **ProgramDriver** and **SingletonTester** supports testing of the singleton. These object-oriented principles ensure efficient, scalable, and modular design that meets the software'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** | MacOS has strong support for web technologies, especially for development. It is suitable for hosting web apps with minimal configuration. | Linux is highly efficient for web hosting. It is free and open source, providing more control over server setup. | Windows provides robust support for web hosting with IIS and is ideal for teams already familiar with Windows environments. | Mobile devices have limited support for hosting, so a cloud-based server is recommended for backend services. |
| **Client Side** | Macs are popular among developers, with excellent support for web browsers and development tools. | Linux requires specific setups and expertise, which can be a disadvantage for some developers. | Windows is widely used, and web browsers work seamlessly. Development tools are widely available. | Mobile devices are ideal for accessing the game. The web app must be responsive and mobile-friendly. |
| **Development Tools** | Xcode is available for iOS development, and development tools for web apps are readily available. Licensing costs for development tools (Xcode) are low, but macOS hardware can be expensive. | Linux supports a wide variety of web development tools and programming languages, such as Node.js and Python. Since Linux is open source, there are no licensing costs for tools, but expertise in Linux setup is required. | Windows supports popular web development frameworks and tools, like Visual Studio and IIS for hosting. Visual Studio is one of the most powerful IDEs for full-stack development, but it comes with licensing costs. | Mobile web browsers must support responsive design for optimal play. Tools like React Native or Flutter could also be useful for mobile-specific features. |

## 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**: I recommend using **Linux** for the server-side platform due to its cost-effectiveness, scalability, and robust support for web hosting. It allows for efficient management of multiple web servers and can be deployed in cloud environments, making it ideal for hosting a multiplayer game.
2. **Operating Systems Architectures**: Linux offers a *monolithic kernel architecture*, providing greater control over system resources. This architecture will allow for fine-tuned performance when hosting the game and scaling as required.
3. **Storage Management**: A **relational database management system (RDBMS)**, such as **MySQL** or **PostgreSQL**, will be ideal for storing game state, player data, and game progress. This ensures data integrity and supports complex queries needed for the game’s functionality.
4. **Memory Management**: Linux uses **virtual memory** techniques to manage large-scale applications. This is crucial for managing the state of the game and handling multiple teams and players simultaneously without running into memory constraints.
5. **Distributed Systems and Networks**: The game will operate in a **client-server architecture** where the backend (hosted on Linux) will communicate with clients via HTTP requests. This distributed system will rely on web APIs, and the network must be optimized for low-latency communication to ensure smooth gameplay.
6. **Security**: To secure user data, **SSL/TLS encryption** should be implemented for all communication between the client and server. Additionally, user authentication mechanisms such as OAuth or JWT should be used to secure player accounts and game data. Proper server-side validation and encryption will protect against unauthorized access and ensure privacy.