

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 | 05/23/24 | Jesse Moore | In-depth look at important information for the client. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has approached us here at Creative Technology Solutions (CTS) in order to develop a web-based version of their current Android game, Draw It or Lose It. This game, similar to the 1980s television game Win, Lose or Draw, involves teams competing to guess images drawn from a stock library. The web-based application must support multiple platforms, allow for multiple teams and players, ensure unique game and team names, and manage a single game instance in memory at any given time. This proposed solution will address their requirements and streamline the development process.

## Requirements

According to the client, this software must support the creation of multiple teams with multiple players on each team, the team names must be unique, and there must only be one instance of the game in memory at any given time.

## [Design Constraints](#_2et92p0)

Design constraints include ensuring cross-platform compatibility, maintaining a consistent user experience across varied devices, handling real-time data synchronization between clients, and managing the server load efficiently. Additionally, security measures should implemented to protect user data across various platforms.

## [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 provided UML diagram below outlines the structure of the requested game application. Key classes include Game, Team, Player, and Entity. The Entity class serves as a base class, encapsulating common attributes and behaviors shared by the other classes. The Game class manages game logic, while the Team class handles team-specific functions. The Player class manages individual player data. Object-oriented programming principles (inheritance, encapsulation, and polymorphism) are demonstrated in the diagram, allowing for a modular and scalable design that helps to efficiently meet the software 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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers offer excellent performance but can be expensive | Linux servers are reliable and cost-effective, widely used in server environments. | Windows servers are user-friendly and offer good performance. | Mobile devices are not typically used as servers, limited resources and performance constraints. |
| **Client Side** | Developing for Mac can be costly and require specific expertise in macOS development. | Linux client development is cost-effective but may require specialized knowledge and tools. | Windows client development is widely supported but must consider licensing costs. | Developing for mobile devices requires expertise in iOS and Android development, which can be costly and time-consuming. |
| **Development Tools** | Relevant tools for Mac include Xcode for development and Swift/Objective-C for programming. | Linux development tools include GCC, Eclipse, and languages such as C++, Python, and Java. | Windows development tools can include Visual Studio and languages such as C#, .NET, and C++. | Mobile development tools include Android Studio for Android and Xcode for iOS, using languages such as Kotlin, Java, Swift, and Objective-C. |

## 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 as the operating platform for the server side due to its cost-effectiveness, reliability, and performance. When it comes to client-side development, supporting multiple platforms will ensure broad accessibility.
2. **Operating Systems Architectures**: The chosen operating platform architectures should include a Linux-based server architecture and cross-platform client architectures, utilizing web technologies such as HTML5, CSS3, and JavaScript for broad compatibility.
3. **Storage Management**: A relational database management system (RDBMS) such as MySQL or PostgreSQL is recommended for managing game data, ensuring data integrity and efficient querying capabilities.
4. **Memory Management**: The game should only have one instance in memory at any given time. The recommended operating platform should utilize efficient memory management techniques, such as garbage collection for Java-based applications and manual memory management for C++ components, to ensure optimal performance.
5. **Distributed Systems and Networks**: Network connectivity, data consistency, and system reliability must be accounted for. To enable communication between various platforms, the application should leverage proper software for data exchange and real-time communication.
6. **Security**: Implementing measures for secure communication, data encryption, and regular security audits will protect user information. Additionally, employing two-factor authentication and authorization methods will further help to safeguard user data.