

<Name-of-Software-Application>

# **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 | <01/27/24> | Joshua Hunter | Changes were made to the executive summary, design constraints, domain model, evaluation, cover page and recommendation. |

**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)

This project for is for Creative Technology Solutions (CTS), who wants to make a web based game Draw It or Lose It. The goal is to expand the current Android app to a cross-platform web app that retains the feeling of the game while introducing new features. The main goal is to allow multiple teams, with multiple players, ensure uniqueness of the game and team names, and restrict the existence of only one game instance at any given time.

## Requirements

They want the game to support multiple teams and multiple players, they want the game and team names to be unique, and only one instance of the game can exist in memory simultaneously.

## [Design Constraints](#_2et92p0)

This would require server software that can manage multiple teams and players and games while ensuring their names and ids are unique, it should be written in java.

It should support scalability to accommodate potential growth in user base and game complexity

There should be some platform independence to allow users to play seamlessly across various devices

It should be able to work on both desktop and mobile common browsers

## [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 main part in the ProgramDriver Class sets up the game with specific info. To make sure we're only getting one game instance, the ProgramDriver Class checks with the SingletonTester Class. The GameService Class is like a box for the one-game rule and acts as a big boss. The Game Class builds on the GameService Class and includes all the team names from the Team Class. Players' unique info is added to the Team Class through the Player Class. Also, the Entity Class holds common stuff shared by the Game, Team, and Player Classes.

**"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 serves as a Unix-based operating system, ensuring a stable and secure environment for hosting web-based software applications. While integrating seamlessly with Apple technologies and providing robust security features, Mac may have limitations in hardware options and is less commonly used in server environments. | For a cost-effective and efficient solution, Linux stands out with its highly customizable and open-source nature. Though there's a learning curve for Linux newcomers, the extensive community support, stability, and performance advantages make it a popular choice. | Windows supports technologies like Internet Information Services (IIS) and languages such as ASP.NET. Its seamless integration with Microsoft technologies and user-friendly interface make it suitable for applications. the potential for higher licensing costs and historical security perceptions compared to Linux are factors to consider. | Mobile devices present unique challenges for hosting web-based software applications due to limited processing power and memory. With platforms like iOS and Android, each having its development environment, mobile devices offer direct access to device-specific features. However, they face constraints in resources compared to traditional servers. |
| **Client Side** | Supporting multiple types of clients on Mac involve potential higher upfront costs for development tools due to hardware requirements. Expertise in macOS and iOS development is essential. | Supporting multiple client types on Linux requires consideration of cost-effective open-source tools. However, there is a learning curve for developers unfamiliar with Linux, and expertise in Linux development tools and libraries is crucial. | The considerations include potential licensing costs for development tools, efficient development using Visual Studio, and adapting to Windows guidelines, which might take time. Expertise in Windows development and possibly the .NET framework is necessary. | Developing for mobile devices involves considering costs, with iOS development potentially requiring a Mac. Dual-platform development may take longer, and expertise in platform-specific development tools like Xcode and Android Studio is necessary. |
| **Development Tools** | Languages and tools for Mac development include Swift and Objective-C, with Xcode being the primary IDE. | Linux supports various languages such as C, C++, Python, and PHP, with IDEs like Visual Studio Code, Eclipse, or command-line tools. | For Windows development, languages like C#, VB.NET, and others are used, with Visual Studio as the primary IDE. | Developing for mobile devices involves using Swift and Objective-C for iOS, Kotlin and Java for Android, and IDEs like Xcode and Android Studio. |

## 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 would recommend Linux as the operating platform for Draw It or Lose It to support expansion to other computing environments. Linux offers a robust, open-source environment known for stability and security. Its wide adoption ensures compatibility across various devices.
2. **Operating Systems Architectures**: The Linux setup has three parts: the kernel, hardware layer, and application layer. For Draw It or Lose It, the kernel is part of the system and doesn't need changes. The hardware layer handles the computer's storage and memory but won't need messing with once it's set up. Lastly, the application layer holds the game's server application. It deals with memory, storage, and talks to the players.
3. **Storage Management**: I would use a cloud-based storage system compatible with Linux, such as Google Cloud Storage for scalable and reliable storage. Cloud storage ensures seamless data access across various devices while leveraging the benefits of a Linux environment.
4. **Memory Management**: Linux is a flexible and straightforward operating system. In this game, we'll be using Java as the primary language for backend development. Java takes care of memory management on its own through a system called a garbage collector. So, there's no need for us to write memory management code in our application.
5. **Distributed Systems and Networks**: To improve Draw It or Lose It, use multiple servers in different places for speed and reliability. Share the image database among servers and clients to save space. Keep user accounts in a central place for data consistency across servers.
6. **Security**: Using Google Cloud makes things easy. Google takes care of the hardware and security, and we use Google apps for the software. The game gets automatic updates, and we check it before releasing it.