

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 9/26/2024 | Francis Torres Rios | Preparing software design document |

**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](#_heading=h.35nkun2)

The Gaming Room presently offers an Android app called *Draw It or Lose It*. They are looking to expand by creating a web-based version of the game that can run across multiple platforms. This game involves several different games, with each game hosting multiple teams, and each team consisting of several players. To ensure smooth gameplay, both game names and team names must be unique. *Draw It or Lose It* features an extensive collection of pre-made drawings. However, the team at The Gaming Room lacks the knowledge to set up the required development environment.

## Requirements

One or more teams involved, each with multiple players.  
Game and team names must be unique.   
Only one instance of the game must exist.

## [Design Constraints](#_heading=h.1ksv4uv)

Android, iOS, and the web each utilize distinct software development kits. The API should be designed to function seamlessly across all three platforms. It must also support one or more teams from any of these platforms. Both game and team names need to be unique. The system should notify the team captain if a chosen team name already exists, allowing them to select a different one. Unique IDs should be assigned to each game, team, and player to ensure only one instance of a game is running at a time.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

The ProgramDriver class houses the main method. It uses a directed association with the SingletonTester class to check if an instance of GameService already exists. The Entity class acts as a parent to the Game, Team, and Player classes, with all three inheriting the necessary attributes from Entity. A player cannot belong to multiple teams, but a team can have players. Similarly, a team cannot own a game, but a game can include teams. While a game cannot possess a GameService, the GameService can manage games. There must be only one instance of each game running at any given time, with each game having only one unique team, and each team having only one unique player.

**"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](#_heading=h.z337ya)

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** | Mac offers strong performance and stability for hosting web applications, especially when combined with Unix-based tools. However, it's generally more expensive and less commonly used in server environments compared to Linux or Windows. | Linux is widely known for its flexibility and cost-effectiveness in hosting web applications. It supports a variety of web servers, but may require advanced technical expertise to set up and maintain. | Windows servers are easy to configure and integrate well with Microsoft software. However, they are generally more expensive than Linux solutions and may have compatibility issues with some open-source tools. | Hosting web-based applications directly on mobile devices is uncommon. Instead, mobile devices access applications hosted elsewhere, but considerations include optimizing for different operating systems like Android and iOS. |
| **Client Side** | Mac development focuses on supporting both desktop and mobile clients, with strong integration for iOS applications. Costs and expertise required are relatively high due to the need for specialized Apple development tools. | Linux development requires attention to the various distributions, each with its own quirks for client support. Cost is low, but development times may be longer due to the fragmented nature of Linux client environments. | Developing for Windows clients is straightforward, with tools like Visual Studio providing good support. However, ensuring compatibility with older versions of Windows can introduce extra complexity and time. | Mobile development needs to account for a variety of screen sizes and hardware capabilities. Expertise in mobile-specific languages and frameworks like Swift or Kotlin is crucial, but costs can increase due to the variety of devices. |
| **Development Tools** | Mac development commonly uses Xcode and Swift for native applications. Web-based apps can be built using JavaScript frameworks like React or Vue.js and optimized for Safari. | Popular development tools on Linux include text editors like Vim or IDEs like Eclipse, along with languages like Python, Java, and Node.js. The flexibility of Linux makes it ideal for a wide range of programming languages and tools. | Windows supports a range of development environments, with Visual Studio being a key player. Common languages include C#, .NET, and JavaScript frameworks, making it suitable for web and desktop app development. | Developing for mobile devices requires specialized tools like Android Studio for Android and Xcode for iOS. Cross-platform frameworks like Flutter and React Native can also be used to build apps for multiple devices simultaneously. |

## 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**: My suggestion is to use a Linux Ubuntu Server on an AWS cloud setup.
2. **Operating Systems Architectures**: The Linux kernel is secure, reliable, and stable. AWS cloud offers easier integration services.
3. **Storage Management**: For storage I recommend using SSD as it is miles faster than HDD, but HDD also works if there are budget limitations.
4. **Memory Management**: I recommend setting a fluid load system where the required memory alternates whether there are more players or not.
5. **Distributed Systems and Networks**: Since this system will be cloud-based, any maintenance required on the servers won't disrupt the game. You can simply migrate the node to another server, and if the system crashes, a new server will automatically take over. This approach enables you to host everything necessary for running the game, aside from the client side. It also allows for the creation of clients for all operating systems to access the game's data. The separation of functions provided by AWS simplifies the management and organization of your system.
6. **Security**: Role based security system such as admin, team, and player.