\

Draw It or Lose It

# **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 | 09/17/23 | Tyler Mort | Added executive summary, requirements, design constraints, system architecture view, domain model, evaluation, and recommendations |
| 1.1 | 09/30/23 | Tyler Mort | Revised evaluations |
| 1.2 | 10/15/23 | Tyler Mort | Revised and clarified recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The client would like a web-based gaming app which will steadily complete a drawing over 30 seconds for a team to guess. The drawings are different for each team and failure to guess correctly during the timeframe will give 15 seconds for other teams to steal. This requires a runtime environment that con simultaneously handle several unique users at a time and attribute points accordingly. This can be handled through a singleton class structure and array lists to store point values as well as several other Java components.

## Requirements

* More than one team available
* Each team may have multiple players
* Both game and team names must be unique
* Only one game instance can run at a given time

## [Design Constraints](#_2et92p0)

For this web-based client to work, it will require connection to a server process in order to ensure smooth operation for all users and limit the prevalence of bad actors. This will allow web users to visit the web application and enter the appropriate player name and join a game. The game will run on the back end of the server for all players connected to the same game. This will allow the program to distribute images properly to each player and award points accordingly. Since each player must be unique for point keeping purposes, the program will check if the player exists before creating their profile. It will also do this for each team or game created to avoid any overlap and therefore possible data races.

## [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)

In this diagram, the ProgramDriver uses the SingletonTester as part of its main method. The separate class Entity is the parent class to the Game, Team, and Player classes. As such, these classes inherit attributes and methods from the Entity class. The GameService class interacts with the Game class in a zero to many relationship. The Game class interacts with team in a zero to many relationship. And the Team class interacts with Player in a zero to many relationship. These relationships allow the GameService class to interact with the other classes to ensure unique identifiers.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | A non-standard option for hosting server-side data. As such, the costs are likely greater due to lower usage and demand. Licensing costs will greatly increase with scale. REST API is independent from platform on both server and client. | Heavily open sourced with many utilities, distros, developmental libraries. The many different distros lead to a huge variance in power, scalability, and security, however, there are very powerful distros available. Open source nature provides many readily available solutions to problems. REST API is independent from platform on both server and client. | Microsoft web framework solutions are robust, however, they can get expensive for licensing. Easy to use user interface with customization, however, that can often hurt performance. Widespread compatibility for applications. REST API is independent from platform on both server and client. | Increased prevalence of mobile devices has increased support in recent years. Requires conversion between mobile and computer systems as there is not a native connection as of yet. Many options for data conversion exists, however, it can cause worse performance. REST API is independent from platform on both server and client. |
| **Client Side** | There are several APIs for interactions with Safari. This allows the server to communicate with the user on Safari or Google chrome with minimal programming effort. | Lack of dedicated support. Due to the segmented distros, there are fewer to no official channels for support. This may make it more difficult for the client to troubleshoot Linux specific problems. | Supports all major web browsers natively. Improved optimization for chromium based applications like Edge. Widespread usage makes it easier for user troubleshooting. | Lower processing power on device due to lack of dedicated hardware such as a GPU or CPU. The reliance on a small APU results in worse performance and therefore a greater need for program refinement and optimization. |
| **Development Tools** | Many developmental tools have expanded to work on all OSes. IDEs such as Eclipse can be installed and used regardless of OS due to several versions. The Mac uses a standard multi-threaded architecture. REST API is independent of hardware or programming language. | Many tools have expanded to work on Linux distros. IDEs such as Eclipse can be installed and used regardless of OS due to several versions. Linux uses a multi-threaded architecture; however, it sometimes struggles with languages like Python that work on interpretation. REST API is independent of hardware or programming language. | Most if not all major languages are supported for Windows development. Focused on .NET and C# which are natively packed into the Windows distribution. Java, Python, and more are also supported with relevant libraries installed. REST API is independent of hardware or programming language. | iOS and Android have very different approaches to applications. Android supports much wider deployability through APKs. iOS requires more reviews before publishing which can be costly and restrictive. Both take large cuts of profit for officially released programs on their store. |

## 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**: The Linux operating platform will allow for a widely scalable web application. While The Gaming Room could produce their own servers, they can be quite costly. As such, the usage of the cloud through programs such as Scala or Azure can reduce costs by utilizing already established servers. These servers often charge per API call or by the amount of hardware used such as storage, bandwidth, CPU, and RAM.
2. **Operating Systems Architectures**: The Linux operating system is split into the Kernel, which virtualizes and manages the hardware directly, the Shell which acts as an interface between the user and the Kernel, and the application layer which directly interfaces with the user. This allows the OS to prevent malicious attacks to sensitive data and hardware by restricting access to the Kernel. Due to the open-source nature, there are constant updates to increase Kernel security and prevent malicious actors.
3. **Storage Management**: Cloud servers use large pools of hard drives and tapes. The data is sent in an encrypted manner to allocate subsections dedicated to the Linux program set to run. The server then can access the memory again using its allocation location to send back to the client. This system requires an overhead program to manage and run the drives. This creates great redundancy and security due to constant encryption and data being saved to backups as well.
4. **Memory Management**: Linux supports both physical and virtual memory natively. Since cloud servers often charge based on the amount of RAM needed, it is important that the program is optimized to reduce costs. This can be done by utilizing virtual memory to reduce the dependency on physical memory and utilize storage space as needed. It is important that less frequently used objects and processes are closed in order to minimize memory usage. This can be done through embedded memory management protocols and subprocesses.
5. **Distributed Systems and Networks**: Linux is natively supported on Windows through the inclusion of Windows Subsystem for Linux or WSL. This means that the app can be accessed with no modification on more platforms. Cloud systems balance usage between different applications to minimize and distribute the load on the system. This active switching process reduces the total demand on the hardware and helps to keep all programs on the server running constantly. Servers also run backup services for a small fee which allows for the protection of data in case of a server outage. This also allows for minimum downtime while a server is taken down for maintenance and then redeployed.
6. **Security**: The usage of an account with a username and password is common for web-based applications. They provide a level of security that is minimally invasive and has low impact on user experience. By checking an account’s validity before accessing the game or server, it helps prevent bad actors. The program can also check the cookies of the client when connecting to the server to better detect malicious attacks. Ensuring that updates are required on a consistent basis, as well as periodically verifying files will help to prevent easily targetable weaknesses in the program.