

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.1 | 2/18/2022 | Jesse Jayne | Completed Recommendations section. |

**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)

The Gaming Room has hired CTS to streamline the development of the web-based version of their Android application, Draw It or Lose It. The game requires that the software allow for multiple teams and players, all with a unique name or identifier, and that only one instance of the game can exist in memory at any given time. CTS will accomplish this through object-oriented design and programming in Java for the web-application, creating a base class for all entities to inherit a unique identifier from. The program will check these identifiers against all other entities to ensure uniqueness among other users, teams, games, and ensure a single instance of all at any time.

## [Design Constraints](#_2et92p0)

Creating the web-based version of Draw It or Lose It, from the Android version, will present several constraints.   
1. Moving from a mobile platform to a web-based one will require a change in programming language and approach. Java will be used to maximize modularity of classes and object-oriented design, as well as compatibility with the largest number of web platforms and clients.

2. Moving into the web, client security and performance will be at a higher risk and premium. Ensuring that each player, team, and instance of each game is unique is paramount. Thus, the coding approach of using a singleton design pattern to check and constantly ensure memory fidelity will be used. Again, Java will excel at this application.

## [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 diagram below shows the high level overview of the planned code for the application. Beginning with the foundation of the base class, Entity, all classes will inherit the id and name features to ensure they are unique. The GameService entry will run alongside the children classes, Game, Team, and Player, in order to assist in accessing and modifying the Game class, in order to add new games, get information about what game is being played, and holds the singleton pattern that maintains the single instance of a game running at a time for memory and performance efficiency. ProgramDriver, the main file, will use the SingletonTester file in order to check for the single instance and unique identifiers of all entities while running the application.

**"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** | Mac web hosting is famous for it’s secure and top of the line hardware. Safe and fast, but highly expensive to host on. Recommended. | Linux also runs quickly for hosting, but will require more work for security. However, it is much cheaper than Mac. | A must if the application uses and Microsoft proprietary integrations, not recommended otherwise. Slow. | Unusable as server/host, and highly difficult to use as a base for communicating with other platforms. |
| **Client Side** | If making a desktop application, requires more time and development, different languages, but can run web/browser based applications easily, and securely. | If making a desktop application, requires more time and development, different languages, but can run web/browser based applications easily, and securely. | Very easy to develop desktop applications, but difficult to get them to communicate with users on other platforms. | Developing from Android to iOS can be difficult and requires different languages and software, and expertise. Running the application in-browser on mobile devices may require drastic quality and performance cuts. |
| **Development Tools** | Modern MacOS devices may require Swift programming language for desktop app development. Java can be used in Swift with the right packages and libraries, but requires time and expertise with Mac developers. | Java can be used natively for developing applications for Linux, easily used and maintained on this platform. Eclipse java IDE works on Linux. | Java can be used natively for developing applications for Windows. Some C or C# may be needed depending on Windows integrations or hosting. Eclipse IDE works on windows. | Each mobile platform will require different language and IDE usage and hosting. iOS will require Swift, but can take a lot from the work done on the desktop app, perhaps a good idea to port the app to iOS after the web-based version is complete. |

## 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 recommended operating platform is MacOS. The safest and fastest hosting, and with better communication with the desktop application, MacOS will always ensure quality and security. The cost will be higher but the ability to then port the application to all other systems, including mobile applications on the iOS platforms is a huge bonus.
2. **Operating Systems Architectures**: MacOS incorporates layered design to assist in rapid development of applications. Along with Apple’s IDE Xcode which may assist with the Java/Swift communication, the development will be relatively easy although slightly slower and more costly than other options. The iOS platform being a version of MacOS allows for easy communication and workability between platforms.
3. **Storage Management**: Utilizing a cloud storage option, AWS being recommended, will ensure the best storage across all platforms and best communication with the same storage being able to be used regardless of device or platform.
4. **Memory Management**: If hosting the game on a MacOS device or server from which all users will be accessing the game from their mobile device or browser, then there will be a considerable need for large amounts of memory and memory management to account for many users constantly accessing the data. Developing the program using MacOS and then hosting all client accessible data on the cloud platform will move memory management responsibilities to the cloud service and take cost and work away from the development team. To efficiently allocate resources, the cloud system can serve data or run new instances of games when they are directly needed, then stop connection, rather than partitioned data on a 1st party server being idle and running constantly.
5. **Distributed Systems and Networks**: Being that the game will run mainly on web browser and mobile app platforms, the connectivity is relatively straightforward, as the users will be on an internet connection to play the game. The teams or connections between the players can be either peer-to-peer or Client-server based connections. For the obvious security reasons and for complexity of scale and resources, Client-server is much more highly recommended. The server, cloud based preferably, holds all game logic and game state, and all communication between players goes through the game server and never directly.
6. **Security**: As the previous sections have stated, every decision recommended throughout the development and implementation process for this project has been made with security in mind. The operating platform is the safest available, with Apple having security as a top priority, and the ability to easily add verification into any code or applications accessed by or from a MacOS device. Deciding to use the mobile and web-based platforms also allows for the inherent security of the client-server connection, wherein no user will be able to access any other user’s data or connect with them directly. A web browser or mobile app game can also require a log-in or two factor authentication before starting, improving security immensely. Lastly, as this game is a relatively quick game running a single instance from the cloud and then disconnecting, there is not much need for saving or accessing user data, and none will be saved or exposed back onto the cloud game server.