

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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)4

[**Recommendations**](#_m8aleynsvzvc)8

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/22/2021 | Charles French | Added Executive Summary, Design Constraints, and Domain Model contents to template. |
| 1.1 | 6/6/2021 | Charles French | Added Evaluations. Updated page numbers in Table of Contents |
| 1.2 | 6/19/2021 | Charles French | Updated 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 wants Creative Technology Solutions to develop a web-based platform for their currently Android only game, Draw It or Lose It. The software must be able to run multiple games, which have one or more teams involved. The teams must then have multiple players assigned. Further, game and team names should be unique, and allow users to see if a given name is in use when choosing a name.

This can be accomplished using one system of collections that will assign unique id numbers to each game, team, and player. With the combination of names tied to unique ids, the system of collections can be searched for the presence (or absence) of a particular name. For instance, Sally Gamer creates the team “Blue” while in the game named “Party Time!”. The game is checked for instances of the team name “Blue”, and if it doesn’t already exist, it is assigned an id number and is created. However, in this case, Joe Player already created team “Blue” before Sally. Thus, Sally is informed that that team already exists, and can be prompted either to join team “Blue” or use another name for her team.

## [Design Constraints](#_2et92p0)

Being a web-based environment, depending on the amount of concurrent users, the server could get overloading with creating many games, teams, and players, so there may need to be a maximum number of games allowed to run at once. This ensures that each generated game provides the user the intended experience. Also, being over the internet, there are possibilities that users may be disconnected from the service during a game, so the game must have a way to account for dropped or unresponsive users. For instance, if a user is unresponsive for several minutes in a round in the game, the game might remove that player from the game so that it may continue to play for other users. Lastly, there may be need to build the system to stave off attacks such as DDoS. This could be done through requiring a player to log in to an account before joining a game, or by implementing features such as reCAPTCHA to prevent malicious bots from trying to overload the server with requests to create games.

## [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)

First, we have the base class Entity, that contains the majority of the structure for its 3 child classes, Game, Team, and Player. This shows both inheritance and polymorphism, as the child classes implement additional features and modify the toString method from the Entity class. The GameService, Game, Team, and Player classes have a zero to many multiplicity relationship. That is, GameService can function with any number of Game instances, Game can function with any number of Team instances, and Team can function with any number of Player instances. The relation is due to the ArrayLists contained in the GameService, Game, and Team classes, as they work together to form collections of one another, with Player class instances being the smallest unit. While there is need for multiple instances of Game, Team, and Player classes, we need one and only one instance of the GameService class to run at any given time, which is why the diagram shows the class containing the singleton pattern (private constructor and creation of a private instance of itself, with a public “getter” function to retrieve the one instance of itself). By having only one instance of the class with the largest “container”, that is, the ArrayList of Game classes, which in turn have ArrayLists of Team classes, that also have ArrayLists of Player classes, we are able to catalogue each instance contained therein with a unique id number, fulfilling the requirements of having multiple teams in one game, having multiple players in a team, allowing the program to check for game and team name uniqueness, and allow only one instance of any one game in memory at one time.

Lastly, we have the ProgramDriver and SingletonTester classes. As simple as it is shown, the ProgramDriver has a relation to the SingletonTester class because it uses SingletonTester to make a call to test functionality of other classes in the package.

****

## [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** | With the proprietary hardware, Mac would be one of the most expensive of the three to start hosting. Further, lack of ability to really update the hardware could be troubling for the server in the future. However, good vendor support would assist the server admin in times of need, and the ease of use of the OS would allow for quick deployment. | Linux makes it easy to add and configure hardware, so if the server needed to increase its storage it could do so with little trouble. The OS itself is highly configurable and could be tailored for the sole purpose of running the web app. Issues could include the lack of driver compatibility, especially if the server is built with older hardware. Lack of vendor support would also mean that it would be up to the server admin to hopefully find a wiki page or forum post online about issues had. | Windows has had server specific operating system software for a while now, so deploying the web app would be a quick process, with little worry about updates bringing the server down (can be easily configured to plan outage times for updates). Windows is a major target for viruses, though, so a strong network and anti-malware system would need to be set up as well to secure the server from attacks. Support from the vendor would help in times of need, though the licensing costs for the software would be significant. | While being a target device for the client, mobile devices would be a poor choice to host a server on due to the lack of power in their hardware compared to the possibilities of the other three options. OS updates to the device would not be configurable and would happen as they are deployed, causing outages and potentially, the server software may break after the update. |
| **Client Side** | Unless the client software is made to run in the web browser (and thus not appear in the App Store, losing a lot of visibility), the program would need to be written in either Objective-C or Swift to be compatible with the device. While the other three platforms could run with mostly the same code if written in Java, making the application extend to this platform would result in essentially doubling the work needed on client development. | Linux would allow for serval programming options and would be similar to writing the program for windows and mobile devices. There may be some differences though (such as focusing on utilizing Vulkan over Window’s DirectX for rendering game content for broader compatibility). Knowing that Linux has well established Windows emulation, though, it may be best to focus resources on polishing the Windows release, then checking compatibility using tools such as WINE to test for any possible performance issues. | Windows would have multiple options available as far as programing languages go. It could be written in Java to save time so that it and other platforms could be written mostly by one set of code, though performance would be improved if written in C++. Given the nature of the program, though, it is unlikely for the program to be that taxing on the system to notice the performance boost. That said, given the large variety of hardware configurations for Windows users, time would have to be taken to see how compatible the program is with older hardware to be able to tell the user what the minimum requirements to run the game are. | For android devices, the software could be written in Java and be able to run on any device, though the performance may vary depending on the device’s hardware. Further, licensing would be something to keep in mind, as the android SDK, available through the IDE Android Studio, would have a cost for each developer on the project. iOS devices would be similar to the issues with MacOS, where they would need to be developed in different languages from the other platforms, adding on to the time needed to release the game on all devices. |
| **Development Tools** | Xcode is the IDE of choice for Mac development, which is part of the Apple Developer Program, Apple’s collection of tools used for developing on their platform. This comes at a cost of 99$ per year per seat but comes with a wide array of support tools such as beta releases of the OS to test compatibility. To utilize the Apple Developer Program, each developer would need to be running on a Mac device. The program would be developed using Objective-C, Swift, or a combination thereof. | To develop on Linux, popular options would include Eclipse, Bluefish, and NetBeans for an IDE. Unlike Mac, Linux apps would not have to be developed on a Linux system, so development could be done on a Windows platform using Microsoft Visual Studio as well, though that option would require a licensing fee. Language options would include Java, C++, and Python, with Java being the lead choice for compatibility. | Top choices for Windows development would include Eclipse if writing in Java, or Microsoft Visual Studio if writing in languages such as C++ or Python. If also writing Java for Android development, an IDE such as Android Studio would allow for development on both platforms at once. For business solutions, though, Android Studio and Visual Studio would involve licensing fees. Windows has broad support for many languages, but top picks would be Java or C++. C++ would be better for performance, but Java would be better for compatibility. | For Android devices, Android Studio would allow access to the Android SDK, after a licensing fee is paid for its business solution. Once the SDK is installed, though, any other popular Java IDE could be used, such as Eclipse, or IntelliJ IDEA. For Android development, the software would be written in Java.  For iOS devices, the Apple Developer Program, along with one of its tools Xcode, would be the go-to for development. Like Mac, the software would need to be written in Objective-C and/or Swift. |

## 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 Java server application should run though Amazon Web Services as this will provide a broad toolkit to maintain the server, give archival options and storage solutions, and provide scalability if the game receives more traffic than expected, allows for development and launching of updates, and is much more affordable than purchasing and maintaining one’s own server hardware.
2. **Operating Systems Architectures**: Amazon Web Services would be a cloud-based model, meaning that hardware and networking is maintained by a third party, allowing for more focus to be placed on updates to the program, rather than maintenance. As for the server software itself, it would a REST structure, passing JSON scripts back and forth using HTTP requests. The server would store the information for each game in its own instance via use of the singleton pattern. This information would then be stored to a file for archival purposes.
3. **Storage Management**: Amazon’s S3 Standard Storage service would be used to store server data, including the database of users, archived previous games, picture assets for the game, and the server software itself. As more user data is created, or as more features are added to the game itself, the storage service will scale to demand. In addition, Amazon’s S3 Glacier service can be used for long term archival, such as server backups in the case that something happens to the live version.
4. **Memory Management**: Amazon’s Lambda service provides scalable memory management by executing code in parallel to each request. In addition, since the server is written in Java Java’s built-in garbage collector will automatically free memory once no longer needed, such as when a game instance closes.
5. **Distributed Systems and Networks**: By using a REST structure, the server would be able to communicate to clients regardless of the device or the language used to develop the client. JSON scripts would be passed back and forth between server and client over HTTP requests. This system depends on the availability of Amazon Web Services’ cloud, as an outage there would disrupt all running games.
6. **Security**: Amazon Web Services has the feature to create accounts for the server as well as policy settings for role-based access control. By practicing the principle of least privilege, the server itself can be kept secure from employees that do not need access to certain features of the platform. Also, the S3 Standard Storage feature has privacy settings that can hide data from the public. The service also has built in capabilities for encryption key generation for users via AWS Key Management Service, which would allow for the client and server to send the JSON scripts encrypted, keeping the information safe during transit. Virtualization via use of the cloud will also work to prevent hardware attacks against the server.