**Software Design Template**

David Smith

Department of Computer Science, Southern New Hampshire University

CS 230: Operating Platforms

Dr. Vivian Lyon

April 24, 2021



Draw It or Lose It Cross-Platform Game Service

# **CS 230 Project Software Design Template**

Version 1.4

## 

## Table of Contents

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

[**Table of Contents**](#_30j0zll)1

[**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)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/19/2021 | David Smith | Original Version |
| 1.1 | 04/04/2021 | David Smith | Updated Operating Platforms Evaluations |
| 1.2 | 04/18/2021 | David Smith | Updated Recommendations |
| 1.3 | 04/24/2021 | David Smith | Final 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](#_sbfa50wo7nsh)

The Gaming Room desires to expand their Android only game, Draw It or Lose It, to become a web-based cross-platform experience. This will add iOS, Windows, Linux, and MacOS clients to reach more customers. To begin this process, we will have to decide whether to make one cross-platform app or an app for each platform. The server application will require dedicated hardware and networking infrastructure to maintain connectivity for all existing and additional clients.

## [Design Constraints](#_2et92p0)

The client application has to either be cross-platform or have a separate application for each platform.

Platforms include Android, iOS, Windows, Linux, and MacOS.

Each client platform has to be able to communicate with the server.

There has to be a server application to handle games, teams, and players, keeping unique identifiers and names for each.

There can be only one instance of the server running at a time.

Communication between clients and server must be fast enough to work within set game time.

## [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 com.gamingroom domain has a super class named Entity that encapsulates an unique identifier and an unique name. Both of these are private and can only be accessed with accessor methods getId and getName. Entity also contains a private default constructor to ensure that an empty instance cannot be created. It contains a public constructor which requires both a name and an identifier. Entity also contains a toString method. All subclasses of Entity will inherit these methods.

The Com.gamingroom domain also includes a GameService class. GameService is a Singleton and makes sure that any game that is created is unique. It encapsulates private fields for a list of Games, nextGameId, nextPlayerId, nextTeamId, and service, which is the instance of itself. GameService’s only constructor is private, which ensures that another instance of itself cannot be created. All of the rest of the methods are public. It contains a getInstance method that creates the GameService instance if it is not already started and returns the current instance if it is. GameService also contains methods addGame with a name parameter, getGame with either a name or id parameter, getGameCount, getNextPlayerId, and getNextTeamId. Because the Singleton contains the next Player, Game, and Team ids in static fields, each of these can be updated easily to maintain unique assignments for each. GameService can have from zero to many instances of Game.

The Game class inherits from Entity and has a private list of Teams. It has a constructor requiring identifier and name. It also has public addTeam method requiring a string and a toString method. A game can have zero to many Teams.

The com.gamingroom domain also includes a Team class. This class inherits from Entity and encapsulates a private list of Players. It has a constructor that requires an identifier and a name, a addPlayer method that requires a name, and a toString method. All of the methods are public. A Team can have zero to many Players.

The Player class is also a subclass of Entity and contains a public constructor that requires an identifier and a name and a toString method.

The ProgramDriver and SingletonTester classes are used to demonstrate how the other classes work together.

Having the superclass Entity removes the necessity of creating the id and name fields inside each of its three subclasses, Game, Team, and Player. It also gives a single constructor that can be explicitly called from each of the subclasses and an explicit default constructor to avoid creating empty instances of the subclasses. Any changes to how the data is initialized can be changed in the superclass instead of each of the subclasses.

****

## [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 does not have a current server only operating system. All features of a standard installation will affect resources available and security of the operating system in a server application. This operating system is only available to run on Apple Mac hardware and is freely upgradeable. The server component of the OS is available for a small fee.  MacOS is based on Unix. The server is open source. MacOS is secure and has user and file access controls. Docker support is only available virtually, and the server environment is difficult to implement. | Linux is the second most used server operating system. It has a lot of security built in and is available for free. Choosing Linux will require someone knowledgeable with the operating system to keep the system updated and secured. Linux can be run in command line only which will reduce necessary resources. There are many valid versions of this OS that are available for free. Linux is based on Unix. Linux server operating systems are open source. Linux is secure and has user and file access controls along with LDAP and ADP support. Docker is supported. A choice will have to be made on which Linux distribution to use. There is a large online community for Linux for support. | Windows Server is the most widely used server operating system. It receives updates as they become available and has good security overall. It will need some extra evaluation at the current time due to a Microsoft breach which could affect security. Windows can be run in a command line only version which will reduce necessary resources. Windows Server 2019 can be run on most PC hardware. Windows Server does require licensing and can be expensive. Windows is secure and has user and file access controls. Windows supports ADP but has limited support for LDAP. A server version will be necessary for security reasons. Windows Server has per user licensing that might need to be addressed. Windows Server has poor open source support. | Reliance on a battery and operating systems (iOS or Android) which weren’t designed to be used as servers will keep these from being acceptable for the server-side application. The lack of hardwired network connections without special adapters and available computing power are also major disadvantages. |
| **Client Side** | As MacOS has a medium market share, availability and expertise of developers may be lower. This could increase cost and time. If a cross-platform application is developed, this could be reduced. Testing will need to be done on Safari which is specific to MacOS. | Linux has a low market share, but standard users are more likely to be more technically experienced. Developers should be around the same levels as MacOS. There are many different browsers available for Linux. Some of the most popular are Firefox, Chrome, Chromium. Others may also need to be tested. | Windows has the highest market share of any desktop operating system. Experienced developers are readily available. Cost and time requirements should be reasonable. Microsoft Edge is the latest built-in browser for Windows. This will need to be tested along with others available for Windows. | Android is the highest market share mobile operating system. This leads to highly experienced and available developers. iOS has higher market share than MacOS, so both of these should be reasonable. Continuing development of the Android app along with developing a separate app for iOS will be necessary. The apps could just be a framework around the web browser interface used for other OSs. |
| **Development Tools** | Using Java in Eclipse, C# in Xamarin, or QT can make a cross-platform solution.  XCode or C++ using Eclipse can be used if a MacOS only app is desired.  Java in Eclipse is available for free and can be used to develop for this environment. Small developer pool. iOS native and Android development available. Small pool of developers. No access to Internet Explorer. Few IDEs available. Support for many different programming languages including Swift and Objective-C. Unix shell scripting is available. | Java in Eclipse, C# in Xamarin, or QT can make a cross-platform solution. C++ in Eclipse can be used to create a standalone app.  Java in Eclipse is available for free and can be used to develop for this environment. Large developer pool. First platform for most new technology. Largest number of available IDEs but not necessarily supported. Large language support including Swift and Objective-C. .Net support, Unix shell scripting, and Powershell are available. | Java in Eclipse, C# in Xamarin, or QT can make a cross-platform solution. C++ in Eclipse or Visual Studio can be used if a Windows only app is desired.  Java in Eclipse is available for free and can be used to develop for this environment. Large set of quality IDEs with support available. Large language support including C#. .Net support is available. Powershell is also available. | Java in Eclipse, C# in Xamarin, or QT can make a cross-platform solution.  Standalone apps can be created in HTML5.  Java in Eclipse is available for free and can be used to develop for this environment. |

## 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**: Ubuntu Server would make an excellent platform to extend Draw It or Lose It availability to other operating systems. By using Ubuntu Server to host the server part of the system, The Gaming Room can set up the server side of their system on an operating system that can be updated without restarting, has 24/7 support available, and has a large pool of available talent to develop on the platform(Canonical, 2021). If The Gaming Room does not wish to pay extra for Ubuntu Server support subscription, there is a large and knowledgeable online community available for support. It is important to ensure that there are ample resources for the game server and its operating system, especially if it is being used inside a virtual environment, like Docker, Microsoft Hyper-V, or Vmware ESXI. One of these platforms is recommended as a backup of the whole system can be maintained and loaded onto another physical machine or virtual machine to quickly get the system back online.
2. **Operating Systems Architectures**: Ubuntu Server can be used in a command line only mode which will save resources for actual game server operations. The availability of Docker allows the possibility to load separate versions of the game server on the same system. This would allow updating the game server in one Docker container while continuing to serve clients in another. Ubuntu Server is Open Source, which allows for communities and companies to discover bugs and errors in the system and work together to fix them. With file and network access management, Linux ensures that only the users with the right access are allowed to read, write, or execute a file.
3. **Storage Management**: Using a system with a dedicated storage device for the game server and its operating system will ensure that disk seek times are minimized. Having a storage device that is shared with other virtual machines or containers can lead to file corruption, locking of file access by other software, and excessive delays as the system tries to read from multiple sections of the disk at the same time. As Draw It or Lose It will be providing images to the clients, it could be useful to place the actual image files onto a solid-state drive or SSD raid. This would enable fast access to the data. As SSDs become slower at high writes, storing the game server and operating system on a more traditional hard drive or raid array would be more appropriate as they will be updated more often and wear down an SSD more quickly.
4. **Memory Management**: It is important to have enough memory to store the entirety of the image files in memory. This will allow the game server to serve the images to the clients without having to repeatedly copy from storage. To achieve this, all the images should be copied into memory as the game server starts. This allows for one copy from storage operation for each image for the entire time that the game server is running. As Draw It or Lose It currently has 1.6 gigabytes of images, it is important to ensure that memory availability is enough to run all the necessary components of the operating system as well as having a large amount of available memory for the image library to expand. The available spare memory will need to be monitored to ensure that the system is not getting to the point where it will need to start swapping physical memory to the storage device and back again. This puts the system in a worse scenario as it would be if it were constantly loading images from the disk. Instead of loading individual files from the storage to serve to clients, the system would be pushing larger sections of data from the physical memory to the storage and having to pull from different sections of the storage to retrieve single images.
5. **Distributed Systems and Networks**: As this will be a web-based client-server interface, having a fallback network connection to handle network interrupts will avoid user frustration over inability to connect. Also, having more than one system that can take over if the main server system goes down will help keep the platform online. It will be necessary to have a second server keep a copy of the data in the GameService so that it can accommodate this. Having a service like CloudFlare to notify users of maintenance if a data center goes down can provide users with information about when the service will return. Different systems can run individual instances of games as long as they maintain their dependency on the GameService to verify all Games, Teams, and Players have unique identifiers. Having the GameService system in a virtual machine or Docker instance that is backed up often will allow the backup to be loaded on another system and ready to start if the main system goes down. This would be even more helpful if a separate database server with an equivalent backup was used to store client information and game status. This would allow changes since the last GameService system backup to be available to the freshly copied system for serving clients without data loss. The communication layer can be controlled via a REST interface. It would also be useful to consider the size of images that will look best on client devices to determine what types of transformations can be done to the images to reduce the amount of data being sent to the clients. Having a different sized image to be sent to mobile platforms versus desktop and tablet systems could help with bandwidth and data cap usage.
6. **Security**: Using encryption on all communication between clients and the GameService host will protect user information from being exposed through the normal operation of the system. All user data should also be encrypted on any hardware device where they are stored. Security audits and upgrades should be performed on a regular basis to ensure data is not breached. Each Ubuntu Server installation used in this system will need to be monitored and maintained for updates and security patches. Linux’s security features include access control to allow users to only read, write, or execute files that they are permitted. Systems used to host the GameService will also need to be located in a physically secure environment.

References

Canonical.(2021). Ubuntu Advantage for Infrastructure. Ubuntu. https://ubuntu.com/advantage.