

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 | 11/17/2024 | Travis Williamson | Initial addition of Executive Summary, Requirements, Design Constraints, and Domain Model analysis. |
| 1.1 | 11/30/2024 | Travis Williamson | Added Server, Client, and Development Tools details. |
| 1.2 | 12/22/2024 | Travis Williamson | Updated recommendations |

**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 is seeking to have a version of their game Draw It or Lose It developed as a web-based multiplatform title. The game currently exists as an Android app only. The Gaming Room is unsure of the development environment required for the project and has requested that we handle the development of the web-based port of the existing application.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_2et92p0)

During development, presuming we port the existing version directly, effort must be made to ensure that any implementation is platform agnostic. As Java is the primary language for development on Android, the fastest route will be to port the existing version of the application, using LibGDX, a Java based framework for Mobile and Web graphics/game development, in order to decouple the application from any Android specific features. As the web version will live in a distributed and potentially vulnerable location, the application should adhere to OOP/OOD best practices.

In addition, as seen in the requirements, the program should allow only one instance of a game, team, or player to exist in memory at any one time. Games, teams, and players must also have unique names.

## [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 model below shows that for each instance of GameService, 0 to many Game objects exist, for each Game instance, 0 to many Team objects exist, and for each team 0 to many Player objects exist. Additionally, it shows that the Game, Team, and Player classes are all subclasses of the Entity class. This model demonstrates the following object oriented programming principles: Abstraction/Inheritance, the Game, Team, and Player classes extend the base functionality of the Entity superclass. Dynamic Polymorphism, the Game, Team, and Player classes override the toString() method of the Entity superclass. Encapsulation, all of the variables of each class in the model are private and only accessible through the use of Accessor methods.

Through the use of these object oriented programming principles, we are able to keep the code DRY, avoiding unnecessary repetition of code between the Game, Team, and Player classes. We are also able to refactor portions of the previous implementation by replacing Game, Team, and Player objects with Entity objects, only casting to the appropriate subclass as needed.

**"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** | Strengths: Stable, powerful state recovery through time machine, more secure than most other consumer operating systems  Weaknesses: Modest to weak administrative controls, higher cost, limited application compatibility | Strengths: Multiple distributions available, including server specific distros. Fully customizable, potentially high degree of security. Lowest cost, outside of enterprise Linux solutions.  Weaknesses: Limited support, when compared to other operating systems. Degree of security dependent on the distribution. Low application compatibility. | Strengths: Largest user base among the desktop Operating Systems, high application compatibility, large degree of administrative control possible.  Weaknesses: The most common target for malicious actors, not as stable as macOS or Linux\*.    \*Depends on the distribution. | Strengths: The most stable option, the largest user base overall.  Weaknesses: Not scalable, limited options for hosting a server on a mobile device, the options that are present (Simple Server for iOS or a custom Termux/Apache/Ngrok server setup on Android) are limited in their usability. |
| **Client Side** | Cost/Time: Using a single language, in the case of Kotlin Multiplatform, saves on development time and costs.  Expertise: Kotlin is a standard C type language, with features familiar to any developer who is used to programming in Java.  Knowledge of Metal for any graphics rendering. | Cost / Time: Development using Kotlin multiplatform saves on dev time, costs.  Expertise: A familiarity with the Linux environment, Java/Kotlin development, and IntelliJ IDEA.   Knowledge of Vulkan / Proton for any graphics rendering. | Cost / Time: Development using Kotlin multiplatform saves on dev time, costs.  Expertise: Knowledge of the Windows API, IntelliJ IDEA.  Knowledge of Vulkan, OpenGL for graphics rendering | Cost / Time: Development using Kotlin multiplatform saves on dev time, costs.  Expertise (iOS/iPadOS) Knowledge of the iOS SDK useful, primarily handled by Kotlin / the JVM.  Knowledge of OpenGL ES for any graphics rendering. |
| **Development Tools** | Kotlin Multiplatform or Kotlin/Native, using Xcode for the IDE.  Kotlin code can be compiled alongside Java code, as both languages run on the JVM. | Kotlin Multiplatform / IntelliJ IDEA for Linux development.  Kotlin supports Gradle, Maven, and Ant builds. | Kotlin Multiplatform / IntelliJ IDEA for development of the Windows client.  Kotlin code is not natively supported by javadocs, requires use of dokka for documentation generation. | Android Studio or IntelliJ with the Android plugin for Kotlin Multiplatform development on Android devices. Fleet is a newer IDE from Jetbrains, usable for development of both client and server side applications written in Kotlin / Kotlin Multiplatform for Windows, MacOS, Linux, Android, and iOS. Xcode is still recommended by Jetbrains for macOS and iOS. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>  
     
   Debian Linux is well regarded for its security, stability, and is a common choice when hosting a server that requires a minimum amount of downtime. Debian is slower than other distributions to update, with the most frequent updates being security updates and the slowest being feature updates, due to the rigorous stability testing each patch / new feature must go through prior to pushing it to live. This approach to updating ensures that a server running on a Debian system will not encounter stability issues due to new features causing conflicts, bugs, and allows for the operators to have a set update schedule based on the priority of the updates.
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>  
     
   Debian has official support for most modern hardware architectures, including platforms whose hardware is built on Armel (Raspberry Pi v1), armhf (Arm v7), arm64 (Arm v8), i386 (x86 32-bit), amd64 (x86-64), mips64el (Cavium, MIPS, Little-endian 64-bit), ppc64el (POWER8 / 9 systems), and s390x (modern IBM Mainframe) architectures. Debian also has unofficial support for other, usually older, architectures such as sparc64.
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>  
     
   TrueNAS Enterprise storage is an excellent option for storage management in a distributed environment. TrueNAS Scale, available for multiple Linux distributions, brings several powerful storage management features to bear. Built around TrueNAS suite of tools with explicit support for distributed systems, REST APIs, Docker Containers, and Kubernetes among others.
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>  
     
   With Debian, as with other Linux distributions, the kernel handles memory management. The server side application lives in the user space of the system’s memory, with the application requiring a minimum amount of kernel level access, as the graphics / audio element should be handled by the client system.
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>  
     
   Debian Linux is well regarded for its long term stability and is frequently used as the operating system of choice for servers. Debian can be installed and run on a variety of systems architectures. Ideally, multiple servers will be maintained, with at least one acting as a back-up server that can handle a set minimum amount of traffic, enabling a user conscious maintenance period, during which users can be alerted about the ongoing outage / maintenance. With the program written in Kotlin / Java, so long as the target architecture of the client system has an available JVM and the program is as error free as possible, the client side of the program should be as stable as the client side system.
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>

The most common method of security is password based user authentication. The system should have a robust method for storing the encrypted user passwords, in such a way that no external actors have access to the passwords in a usable or readable format. Each user should at a minimum have a unique user id, perhaps with an optional public username, to further obfuscate the user’s login credentials, and a unique password of sufficient strength. Any instance of the user’s login credentials being stored should be encrypted.