

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 |
| --- | --- | --- | --- |
| 1.0 | 07/17/2022 | Michael Duteau | Executive summary, design constraints, and domain model have been written. |
| 1.1 | 07/31/2022 | Michael Duteau | Evaluation of development requirements. |
| 1.2 | 08/14/2022 | Michael Duteau | Recommendations for different elements of the application. |

**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 seeks to implement a web-based version of their current Android application, Draw It or Lose It.

## [Design Constraints](#_2et92p0)

***Business Constraints***

Application development will be contingent upon the expected completion date as well as budget. Changes in development should take deadlines and budget into consideration, but adjustments may be required depending on the changes being made.

Deciding how to host the application via the web and how much data usage the application incurs may also affect the budget depending on server requirements and active player base.

***Technical Constraints***

The application will be written using Java language, implementing existing code from the current Android application.

The application development will need to take into consideration the use of different operating systems and web browsers (Chrome, Firefox, Opera, etc.). The application should function similarly on these different browsers so that users will have an optimal experience regardless of their browser choice.

Using a “cloud” server will need to be considered, as these servers are typically used to run web-based applications. These servers can vary in price and functionality, so these options will need to be discussed in order to make the game accessible and runnable via the web.

## [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 displays classes that relate to the specific object they are meant to hold and control. For instance, each player, team, and game hold their own class. These classes also inherit from the Entity class, using certain variables and methods from the Entity class that are used in its inheriting classes demonstrating the Inheritance property of OOP (Object-Oriented Programming). These separate classes also pertain to the concept of Polymorphism, as certain classes will be accessed only with the appropriate function call. For example, a function call of Player type will automatically be associated with the Player class.

The diagram also shows which elements are public vs. private. Encapsulation is an area of OOP that protects certain data from being inappropriately accessed. For instance, each of the GameService, Game, Team, Player, and Entity classes implement private variables and/or methods, meaning that they can only be used within that specific class. This prevents these elements from being unintentionally modified by other classes. This also greatly decreases the codes complexity and makes it more manageable.

Much of this code uses Abstraction, as most of the code regarding the classes will not be viewable or accessible by the user. The main class the user will have a visual representation of is the ProgramDriver class. All other classes perform functions without showing their processes to the user.

**"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** | Due to licensing costs, using Mac as a server is very cost intensive. Especially if there is a need for multiple servers, these costs can become overwhelming. Likewise, development of a server would require the development be on a Mac device. The OS also contains more features than are needed to host a server. Mac OS has innate security features that gives it more defense against malware than Windows, being Unix based. | Being open source, the Linux OS has the least number of constraints when hosting a server. The cost for licensing of a server is also lower than Windows and Mac. Linux, also Unix based, is susceptible to certain types of malware, but this is not common due to the OS’s lack of popularity compared to Windows and Mac. | Potentially the costliest for server hosting compared to other OS. Microsoft uses their own software platform MS-DOS which differs from Unix based operating systems. While not necessarily the most vulnerable to malware, it is arguably the most targeted system by malicious parties. However, it does have a strong service API that is easy to set up and integrates well with Java. | Server hosting is less powerful than other operating systems and needs a connection to a computer to function properly, making it somewhat redundant. However, it has the potential to support other operating systems in terms of deployment. |
| **Client Side** | As stated in server side, requires Mac equipment in order to develop. Contains easy to use software development kits, but requires experience with Swift in order to develop. | C is used in Linux kernel, which differs from C# and C++, so would require experience with that language for development. Has limited file types and needs alteration to support other file types. | Supports all web browsers despite being augmented for its innate web browser. REST API is supported by Microsoft browsers, but requires more experience compared to other operating systems. | Requires some experience with mobile application development. Weak processing capabilities. Most development experience needed to optimize user experience. |
| **Development Tools** | Would require a Mac and knowledge of iCode using Apple’s Swift development software. However, other open-source development applications may be used, such as Eclipse. | IntelliJ is a popular IDE for Linux, using the Java language. However, as stated in client side, experience with C is likely needed to be implemented in some regard. | Visual Studio is likely the most popular IDE. Most commonplace languages are C# and C++, but has the potential to use most popular coding languages. | iOS and Android use different languages, which can be a major constraint. Languages used to develop on Android likely will not be supported on iOS. Would likely need to use C++ or C# in a program that supports exporting to both operating systems. |

## 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**:

Linux OS will run the application using cloud-based means, such that a third party may provide service and maintenance for the program. This option can be very cost-effective when met with the right provider and will vastly reduce the complexity of server maintenance. Using a serverless option like this allows the developers and programmers to deploy a program without paying heed to the underlying infrastructures, which would be managed by the cloud service. These cloud services will take a function or code from the developers when prompted and either find or make a server to run the code (Datadog, 2021).

1. **Operating Systems Architectures**:

Serverless solutions are typically built using modular architecture, coinciding with the use of Linux, which is also built with modular architecture. This modular architecture functions by using separate modules that contain their own components, oftentimes unique, so that the module in question can perform or optimize a specific task. This means that core components of the serverless solution, as well as the OS kernel, are separate from the modules that will be required to run the application. By utilizing this modularization, these core components are abstracted and protected from being accessed by inappropriate parties, increasing the security of the application as well as the OS (Riley).

1. **Storage Management**:

Cloud storage APIs are becoming standard and, because of their security and ease of use, are likely to be a good solution for this application. With cloud storage, data can be managed, sent, and retrieved between both the cloud and a local system. This flexibility can allow multiple developers to access the storage management without needing to utilize the same physical hardware. Likewise, cloud storage can provide many different sizes in terms of how much data is needed to store elements of the application and, in most cases, is very cost-effective depending on how much storage space is needed.

1. **Memory Management**:

Memory should be managed using virtual memory as opposed to physical memory. Virtual memory can deploy and manage processes larger than the physical memory can handle and can also allocate applications appropriately so that they do not affect other running applications. Linux’s system utilizes virtual memory, giving it access to more memory than what the physical memory would initially appear capable of. Memory management in Linux also allows for allocation of virtual memory segments to be protected, so that any code or processes will be safe from being overwritten by another process (Rusling, 1999).

1. **Distributed Systems and Networks**:

The application would benefit from using an API that is able to support multiple clients over multiple different operating systems. REST APIs are lightweight and exchange between clients and servers via HTTP with security. Such services can be beneficial on the user’s end an optimize user experience; with the API having communication back and forth between clients and servers, it will naturally be able to resolve the user’s request to play to a server that is of good connection, preventing the instance of a user being sent to a downed server or otherwise reaching a server that causes connectivity issues. Because REST APIs have the benefit of integrating well with serverless solutions, it’s likely the best API option for the type of OS architecture that has been recommended (IBM Cloud Education, 2021).

1. **Security**:

Implementing the right requirements for authorization and authentication on the user’s end are pertinent to the security of the application and the servers it runs from. User’s will have an account role that has very limited amount of privilege, such that the changes they make may only affect the appropriate details of their account (username, email, password, etc.). These accounts having the least privilege will allow users to play the game and alter their personal account/profile without making changes to the application, nor would they be able to change certain account settings such as an identification number assigned to the account. Accounts are recommended to be accessed and altered using a standard username/password login requiring an email. A further step could be to implement an option two-factor authentication method, such as providing a phone number that a code can be sent to. After signing in using the appropriate username/password, this code will be required to access the user’s account and application to further confirm that the user is the account’s owner. Another possible security measure, for things such as password recover, could be security questions that require personal answers from the user be given when creating the account. For example, if a user wishes to reset their password and/or have forgotten their current password, they will need to answer the question chosen when the account was created with the answer they provided at that time. Such questions would be very general and standard of my applications’ account recovery processes, such as “What is the name of the street you grew up on?”, “What is your grandmother’s maiden name?”, “What is your best friend’s middle name?”, and so on.

References (“Recommendations” section)

Datadog. (2021, June 28). *Datadog*. Serverless Architecture: What It Is & How It Works. Retrieved August 11, 2022, from https://www.datadoghq.com/knowledge-center/serverless-architecture/

IBM Cloud Education. (2021, April 6). *Rest-apis*. IBM. Retrieved August 13, 2022, from https://www.ibm.com/cloud/learn/rest-apis

Riley, J. S. (n.d.). *10 Reasons Why I Prefer Linux Over Windows*. 7 Linux has a modular system design. Retrieved August 11, 2022, from https://www.dsbscience.com/freepubs/linuxoverwindows/node9.html

Rusling, D. A. (1999). *Chapter 3 memory management*. The Linux Documentation Project. Retrieved August 13, 2022, from <https://tldp.org/LDP/tlk/mm/memory.html>