

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 01/22/2023 | Wade Hudgens | The game Draw It or Lose it will be developed |

**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)

Draw It or Lose it is a video game based on Win, Lose Draw. The objective of the game is for players to guess a phrase based on hits given by a library of images. Currently the game is only available on Android and The Gaming Room wants to convert the game to a web-based version so it can easily be played on different platforms.

## 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)

* This version of the game will be web-based.
* No hardware requirements as of now.

## [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)

* SingletonTester is a dependency of ProgramDriver
* Game, Team, and player all inherit Entity
* Instances of GameService have zero or more Game objects
* Instances of Game have zero or more Team Objects
* Instances of Team have zero or more Player Objects

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Pros:  MacOS is very user friendly with an easy to use GUI. Although MacOS is still vulnerable to viruses, it is not as vulnerable as other operating systems.  Cons  MacOS can be very expensive. Apple products are also not easily upgradable. Mac also has a smaller community. | Pros:  No license fee  Reliable  Not demanding on hardware  Cons:  Forced CLI  Unreliable long term support  Updating can be complex | Pros:  Beginner friendly  Constant updates  Widely known  Cons:  More vulnerable to viruses  GUI consumes resources  Expensive | The main advantage of a mobile device is its portability. This is irrelevant when it comes to servers. There are not many cases where I would chose to use a mobile device for hosting a web-based application. |
| **Client Side** | Supporting mac as a client side application can be done with a Mac OS specific technology like SwiftUI and Xcode or can be done with a technology that is multiplatform like Python, Java, or React Native. | Supporting Linux as a client side is mostyle completed with multiplatform technologies like C++, Python, Java, or React Native. | Supporting windows as a client side application can be used with a multiplatform technology or a windows specific technology like .net. | The most popular ways of building mobile apps are as a PWE, React Native, or Swift. The process is slightly different depending mon IOS or Android, but the tech stacks are close to identical. |
| **Development Tools** | MacOS has:  Eclipse  VS Code  Java  Python  Xcode  SwiftUI | Linux has:  Eclipse  VS Code  Java  Python | Windows has:  VS Code  Eclipse  Visual Studio  Java  .NET  Python | It is very difficult to develop on a mobile device as there is no good tools for mobile development. |

Based on the application design requirements, I would recommend React Naive because no complex optimization is needed for the application. React Native is also multiplatform similar to Python and Java but React Native has a much smoother mobile development process than Java or Python.

## 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**: I would recommend a Linux Server as they are lightweight, affordable, and reliable.
2. **Operating Systems Architectures**:

Hardware: CPU, HDD, RAM

Kernel: Communicates directly with the hardware.

Shell: Interface between kernel and user.

1. **Storage Management**: I would recommend SSD as HDD storage because HDD would be a bit too slow for a video game. On the database side, I have a lot of experience with PostgreSQL and find it a reliable database system. There is no one correct answer to this question as there are many valid systems to use, but I am most comfortable with PostgreSQL. It can be used on all of the popular OS, is open source, and has a robust feature set.
2. **Memory Management**: The Linux memory management subsystem is responsible for the memory management. Linux memory management is highly configurable that can be changed with the /proc and sysctl APIs. This management system handles virtual memory, demand paging, and memory allocation for the kernel and user programs.
3. **Distributed Systems and Networks**: There are many different types of ways to handle the network communication of the game. You could use a peer-to-peer method where one player acts as a host and other players connect to their computer. This removes the need for the server to store any data and can be done in a way where a centralized server is not needed at all. I do not like this method for the game. For Draw it or Lose It, I will want the centralized server to hold all the game data and transmit that data to users as needed. This increases the scalability of the application as the security risks of peer-to-peer are mitigated.
4. **Security**: There are multiple security factors you need to consider when developing web applications. Using HTTPS for our server API would solve many of the networking issues by not allowing our data to be intercepted by a third party. In the modern day, it is good to use a third-party authentication service like Google OAuth or Firebase to handle your applications authentication because it is tricky to develop your own authentication. A mistake can cause a big data breach. If you need to develop your own authentication system there are many things to consider: store passwords as salted hashes and not plain text in the database, login throttling, logging, and many more items. A small mistake in any of these can have consequences that can be expensive, which is why it is good to use a third-party service so you can focus your resources on the important aspects of your application.