

Draw it or Lost 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 | 4/14/25 | Peter Donison | Added 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 Company is looking to expand their audience and playability of their game Draw it or Lose it. It is currently only available on the Android operating system, but The Gaming Company would like the game to also be available as a web-based game.

Within it’s development, Draw It or Lose It will need to have the ability to assign unique teams and unique players within those teams, and check if the requested names are already in use. We will also need to ensure that only one instance of the game can exist in memory at any time. This will increase efficiency and reduce errors.

## Requirements

Business Requirements

* Expand audience and game accessibility from Android OS to a web-based application
* Support one or multiple of teams
* Support multiple players on each team
* Game and team names must be unique
* Allow users to check whether a name is in use when choosing a team name

Technical Requirements

* Use the singleton pattern to ensure only one game instance runs on memory at a time
* Create unique identifiers for each instance of a game, team, and player
* The game must be a web-based application and be supported across different browsers and operating systems

## [Design Constraints](#_2et92p0)

* Ensuring team and player names are unique
  + Allowing users to check if a name is already in use
* Ensuring only one instance of the game runs on memory at a time.
  + This makes it easier to store and synchronize data and inputs amongst the multiple people playing the game. The game instance only runs on the server, and the client connect via the web application and sends their user inputs to the server.
* Use across multiple devices
  + Having the game run only an Android devices can be a challenge due to the many different devices using Android OS, but moving this to a web-based game should allow virtually any device with a web browser and internet connectivity to play the game. This substantially increases the number of devices and platforms that can run the game.
* Internet connection
  + This is a game played by multiple people over the internet and has narrow time windows within the rounds. It is important that there is as little latency as possible to ensure the game runs smoothly.
* Ability to handle multiple requests at once.
  + As the core of this game revolves around multi-player functionality, it is important that the game can handle different requests from multiple people or teams at the same time while running smoothly. People might be joining the game, creating a team or player, or providing answer input all at the same 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 solid lines connected to the hollow arrow show that Game, Team, and Player are all subclasses to the Entity parent class. It looks like the subclasses inherit the toString method and the ‘id’ and ‘name’ attributes.

The classes also show association with each other using 0…. meaning that within List<Player> 0 or more objects can exist on the list.

The broken line with the solid black arrow shows dependency, and the direction of the arrow shows that ProgramDriver has a dependency on SingletonTester.

**"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** | Macs are great at what they’re supposed to do, but usually subpar in what they weren’t designed explicitly for. There is not as robust resources on mac as there is on Windows and Linux. Macs are also expensive. | Linux is a great option. It is free in to install making the initial costs cheaper. It also has a robust community and lots of tools for web hosting. It is probably the most secure option as well. | Windows is another great option. It is the most familiar OS which is valuable. It also has robust support for web development. It has an initial cost to install which will increase costs. | Mobile devices are not designed to be web servers, and would do a poor job. A mobile device is designed to be mobile while a server is traditionally designed to be stationary. These purposes do not mesh well here. |
| **Client Side** | MacOS has fewer devices and hardware to account for. Not a huge userbase, but wealthy and likely higher standards. | Even more niche than Mac. Multiple linux distributions to account for. Can be running on virtually any hardware. | Has the most users. This makes it most familiar and most cost effective to develop for. Larger variety of hardware. | Developing an app for a mobile device can vary between Play Store and App Store. Many Android devices and hardware to account for. |
| **Development Tools** | Has great IDE options. I use JetBrains but Xcode and VScode are also well regarded options. | Linux has lots of IDEs available such as Eclipse, VScode, IntelliJ, and Pycharm. | Windows has Visual Studio available for free which is a good IDE, and also has access to many others such as Eclipse, VScode, IntelliJ, and Pycharm. | We would not typically write code on a mobile device, but Java and Javascript both work well mobile devices. Java and Kotlin are used for Android apps, and Swift is the primary language for IOS apps. |

## 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 recommend Linux as the operating platform for the server side of Draw It or Lose It. Linux is free to use, the most secure option,
2. **Operating Systems Architectures**: Linux is typically a monolithic kernel meaning all of the operating system is running as a single program in kernel mode. Any applications then are stored on top of that in the file system. This adds to stability and security as the application and the OS are separated.
3. **Storage Management**: The data related to the pictures will by far the biggest resource by space. If the generated images are in a defined order, then there could be a downloaded queue of the next 5 images. Alternatively, images could be bundled into packs or themes that can allow downloading a variety of images without needing to download all 200. The data from the lists could be stored using MySQL or PostgreSQL which both run natively on Linux and work well with Java.
4. **Memory Management**: Linux can use the hard drive as virtual memory if the RAM is full. It also supports caching to keep recently used data easily accessible. Linux also works will with Java’s the memory management with JVM to be able to handle multiple players and teams.
5. **Distributed Systems and Networks**: To let *Draw It or Lose It* work with users on different devices and platforms the game will use a client-server network. The server will both run the game and store game data. The server will be the central hub that collects, stores, and manages data between itself and the clients. We can use web sockets to maintain the connection between the people playing and the server so that the game updates in real time and players don’t have to refresh the page to see updates.
6. **Security**: We can utilize an HTTPS connection so that all traffic related to the game is encrypted between the server and the clients. We can also have robust login requirements like two-factor authentication or biometrics in order to access or an account. This will increase security and help keep user information safe and prevent fraudulent logins.