

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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)4

[**Recommendations**](#_m8aleynsvzvc)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/23/2021 | Matthew Oliphant | Created initial software design template |
| 1.1 | 06/04/2021 | Matthew Oliphant | Updated design template: Evaluations |
| 1.2 | 06/18/2021 | Matthew Oliphant | Final design template: 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 a successful gaming company that wants to outsource the development of their web-based game, Draw It or Lose It, to our consultant firm. Draw It or Lose it currently exists as an Android app, but we need it to serve multiple platforms. Since the app currently exists as an Android app, its source code is written in Java. The first hurdle is going to be deciding how the app is to be displayed. Web browsers read HTML5, so our Java code needs to have an HTML5 (and possibly CSS) extension, since these are the languages that web-browsers utilize. From there, we’ll have to decide if we want to use server-side or client-side rendering. Either way, once we have a working version of Draw It or Lose It, on a particular web-browser, we then have to tweak the code for each browser that The Gaming Room wants to provide support to (Firefox, Edge, Chrome, Safari, etc.) to ensure no cross browser issues occur. Additionally, The Gaming Room has supplied software requirements for the game application that affect software design, which must be considered before getting into the game specifications.

## [Design Constraints](#_2et92p0)

A web-based distributed environment introduces several design constraints, one of which would be that each web browser operates uniquely, so what works using a certain web-browser might not work while using another. Draw It or Lose It should work on any browser that supports HTML5. Another design constraint that must be considered is, which operating system is supporting the application? The operating system will affect interactions between the systems hardware and the software. The design constraint to only have a single instance of the game in memory at any given time will be solved using the singleton pattern, and from there we can have zero to many teams, with zero to many players on each team. Depending on whether we choose server-side rendering or client-side will also determine how user data gets processed and returned.

## [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 Entity class is the super class to Game, Team, and Player subclasses. Each Entity instance will each have an id and name, two accessor methods to get the id and name, and method to concisely describe the Entity’s information. The default constructor for the Entity class is made private to force an object to belong to an instance of either Game, Team, or Player subclasses. Each subclass also inherits the id and name variables, as well as the methods that are passed from the Entity class.

The Player class is very basic and provides a constructor for a Player. Each instance of a Player contains an id and name, and a method to concisely describe the Player’s information. From there, each Team contains a list of Players. Each instance of a Team contains an id, name, a method to add a player to the team, and a method to concisely describe the Team’s information. Next, each Game contains a list of Teams. Each instance of a Game contains an id, name, a method to add a team to the game, and a method to concisely describe the Game’s information.

Each Game instance will also have access to an additional class called GameService, which benefits Player, Team, and Game objects. The GameService instantiation utilizes the singleton pattern to ensure there is only a single instance. Each GameService object contains a list of Games, the next Game-id, the next Player-id, and the next Team-id, along with an instance of itself. If an attempt is made to create a second instance of GameService, the instance that was originally created will be returned (singleton). Additionally, the GameService instance has a method to add a Game to its list of Games, and multiple accessor methods to get the Game id, get the Game name, get the Game count, get the next Player-id, and get the next Team-id.

Finally, the ProgramDriver class is used to create the GameService instance. Once it is created, games are created. The SingletonTester class then gets called to see if the games are correctly stored/ displayed. This simple demonstration does not show the full capability of adding players and/ or teams, but could be added very easily.

****

## [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** | MacOS is secure as it implements a layered OS. MacOS servers are reliable and can run most software. However, hosts that offer mac servers are rare/ higher cost. Users not familiar with MacOS may not understand how to use Apple apps to communicate with the server. | Very solid and quick and ran on most servers. Linux is open source & very secure. However, not all devices run well with Linux, requiring third party software. A lot of software that’s supported by MacOS or Windows is unsupported by Linux. | Very common OS with device drivers that ensure hardware capability. Servers cost more than Linux and faces more security vulnerability that requires patching. | Most mobile devices OS’s are Linux or Linux-like (BSD). Fast and secure web hosting. Open source or similar. A lot of people use mobile phones so broad customer base. However, Devs can cause apps to be attack prone. |
| **Client Side** | We need multiple MacOS clients to be able to connect simultaneously. AppKit (a GUI interface toolkit from NeXTSTEP) uses client-side decoration when using a NSWindow. | Utilizes the Domain Name System (DNS) to query from DNS servers. Runs using JavaScript which is what most web browsers use. This makes client-side processing very fast. | Utilizes the universal windows platform (UWP), which is an API that uses several languages including JavaScript, and the .Net Framework. Creates API bridges for iOS and Andriod. | Android OS and iOS utilize client-side libraries and tools in their software development kits (SDK). SDKs allow APIs to communicate with libraries to support clients. |
| **Development Tools** | MacOS was written in Objective-C, but provides support for multiple IDEs. In this case, we could use Eclipse/ Java since our source code is in Java.  Mac OS supports multiple languages and has many tools for development. | Linux OS was written in C and support for IDEs has been a little more restricted. Linux does support multiple languages, but the IDE we could use is also Eclipse/ Java. | Windows OS was written in C and supports multiple IDEs. While the .NET framework provides useful features, we’ll use the Java Platform Enterprise Edition (JPEE). We’ll also use Eclipse/ Java. | Android Studio is the official IDE for Android apps, and Xcode is the most popular iOS IDE. Android studio utilizes Java and while Xcode usually supports Swift, we will use Java. |

## 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 the Java Platform. As demonstrated in the Development Tools section of the Evaluation above, Java can be used to create source code for every OS. It is left to each devices OS to make the hardware communicate, so that the client can communicate through Java. I would next recommend that Draw It or Lose It considers how they want to finance/ invest in data management. Due to them being successful with the Android market, I would expect them to do well with each OS they expand to, thus I’d recommend a dedicated hosting platform or to host locally.
2. **Operating Systems Architectures**: The Java Platform architecture consists of the Java programming language itself, Java Development Kit (JDK), and Java Runtime Environment (JRE). The developer utilizes the Java language with the JDK to create applications using the JRE. The Java Virtual Machine is ported to provide hardware and OS independence, and can be used for both client and server applications. The JRE utilizes many libraries, both based and integrated, and can utilize the OS’s server. Finally, the Java Platform must be deployed to allow clients access to (downloading) the app, which allows any client’s system architecture to operate functionally with the app.
3. **Storage Management**: I’d recommend the client-server design pattern regarding storage management. Draw It or Lose It is becoming a large organization so storing all their user’s data is going to continue to be an issue. As mentioned in section 1, I would recommend that The Gaming Room either utilize a hosting platform (VPS or dedicated) or take on the cost of hosting locally. Either way, storing the application and all its components on the server provides great speed and reliability. The clients then access the server, but also stores data on their device as well which relies on each OS to also have storage management capabilities.
4. **Memory Management**: On the client-side, the Java Platform will utilize one or more API(s) to communicate with the OS of the client. The OS then communicates with the hardware to access memory and manage it for the applications use (pull out of secondary storage and into main memory). On the server-side, memory is managed by each server, which operates similarly to PCs but more security-focused. This app will use Java Virtual Memory to communicate with both the client and servers.
5. **Distributed Systems and Networks**: The client system for Draw It or Lose It will include every client and their system’s architecture running the software. The server system is the server(s) that each client communicates to access databases within the server. I would recommend incorporating a distributed server system so that if one server goes down for any reason, the other servers could pick up the slack. A distributed system allows an application to continue to function properly (sometimes slower), rather than not working entirely. Network speeds will determine how quickly data is returned to the client so it’s important for the servers to be connected in the most efficient way, and for the clients to have a strong internet connection so they can access the servers.
6. **Security**: One security measure is through the use of a Secure Socket Layer (SSL), which is a security protocol that authenticates clients and servers using digital certificates. SSL also encrypts and decrypts communication using unique keys to authenticated clients and servers. The other important security protocol is to use a RESTful API to intermediate between client-server communication. Using SSL certification along with a RESTful API allows the users to be authenticated, thus blocks attackers (but users need strong passwords). To further prevent intruders who create a user profile with the application, authorization is another technique used to ensure users cannot access parts of the application they shouldn’t have access to. Finally, it’s important to have procedures in place for when an attacker is trying to slow the system or crack the code by limiting the number of requests a client can make. This limits the possibility of DDoS attacks. The JVM also implements a “sandbox” that prevents the app from doing things without the user’s permission, providing added security to all clients.