

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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](#_heading=h.35nkun2)

This document is intended to outline the design of the web application version of Draw It or Lose It. The Gaming Room’s existing android game that we are adapting to the web. We are hoping that by moving it to the web we will expand the games reach and accessibility by creating an app that can be played across multiple platforms with internet access.

The web app version will retain the core mechanics and features of the Android version, to make sure that those already familiar with the game have a consistent experience. The development will be done using the Java programming language due to its ability to be written once and run on any device with a Java compatible web browser. This is the chief reason that it will allow the game to reach a wider audience.

This process will not be without its challenges of course.

1. Maintaining a similar user experience is chief among our design concerns. The last thing we want is for the existing player base to be unfamiliar with the new web application
2. Scaling up on the back end will become a concern after the release of the web application. We want to be able to account for the growth in the player base.
3. We want to make sure that connections on this new app are secure and safe. We will be implementing robust security measures.
4. We will want to make sure we are managing our memory resources such as images, sounds, and other assets well without compromising on performance.
5. We will be taking care to ensure that the new app will integrate well with The Gaming Room’s existing back end technologies.

By anticipating these risks we are hoping to mitigate them so we successfully deliver what they are looking for in their web application.

## Requirements

1. Team Structure:
   1. The game will have one or more teams
   2. Each team will have multiple players assigned to it
2. Unique Names
   1. game names and team names must be unique
   2. provide a checker to see if names are already in use
3. Instanced
   1. Ensure only one game instance can exist in memory at a given time
   2. unique identifiers for each instance of a game, team, or players
4. Cross-Browser Compatibility
   1. Ensure that all major web browsers can support the application
5. Budget
   1. The project must remain within the budget specified by The Gaming Room

## [Design Constraints](#_heading=h.1ksv4uv)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

1. Compatibility
   1. The game needs to function on multiple web browsers like Chrome, Firefox, Safari, and Edge
2. Performance
   1. The game should retain high performance quickly loading and smooth gameplay
3. Scalability
   1. The game must be designed to scale with the player base without affecting performance or the user experience
4. Unique Identifiers
   1. Each game, team, and player will have a unique identifier and will ensure that only one instance of the game will exist in memory for each player at any given time
5. Synchronicity
   1. Since it is a multiplayer game data shared between players and teams will remain synced
6. Resources
   1. Efficient resource management will be maintained to run the game smoothly
7. Budget
   1. Development will remain within the allocated budget
8. Integration
   1. The game will integrate with whatever existing systems The Gaming Room has for their existing Android app.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

For the Domain Model we can see that the Entity class is acting as an interface for the different entities specifically the Game, Team, and Player Classes. This represents the principle of Abstraction which is the concept of hiding the complex implementation details and showing only what is necessary for the functioning of the object. As an interface it provides an expected way that any child of the entity class will behave with methods that must be defined in the child classes. This also employs the use of inheritance for again the Entity class to the Game, Team, and Player Classes. Which allows those classes to inherit the same methods and attributes of the Entity Classes. We can also use Polymorphism should we need a list of Entities. Since the Entity Class functions as the superclass of the other three child classes it allows us to treat each of them as Entities. All the classes demonstrate the principle of encapsulation by restricting direct access to some of the attributes of the classes and methods used in the classes.

**"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](#_heading=h.z337ya)

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** | Firstly, Macs are very uncommon for servers. They have a high hardware cost and very few server focused features. Not recommended for a server deployment due to high costs. | Linux is a great choice for the serve-side. It is an open-source operating system so costs will be relatively low unless enterprise versions are preferred. It also has excellent scalability for a wide range of needs. | Windows has the advantage of being extremely user-friendly since it is often most user’s first system. Easy to manage and setup and has strong Windows Server editions. However Costs can be significant depending on the licensing model. | Mobile devices are not typically used as servers and they rely heavily on cloud-based backend systems. Ultimately they are not applicable for server platforms and would need a cloud server system that would be based on either Linux or Windows. |
| **Client Side** | Macs are known for their stability and have good support for web browsers including Safari, Chrome, and Firefox. They also boast a strong support network of software development tools specifically for OSX. They do require high hardware costs and will likely need a mac to run them. | Linux Machines have the advantage of being free to use on any hardware, strong security, and have good compatibility for with most modern web browsers. They do have the weakness of not having the best software support but for our purpose an app specifically for Linux client side is great. | Windows is in use widely and has an extensive library of software and development tools. All major web browsers are supported as well as a large user base. It does require making apps with robust security features as its popularity lends it to be more vulnerable to security threats. | Android and iOS are the top mobile operating systems and both have their own development environments. Both have large development support and large user bases. But they also require platform specific tools and codebases. As well as developers with platform specific expertise. |
| **Development Tools** | For much of Mac development the Java is already a great choice as it is designed to run on the Java Virtual Machine (JVM). Specific tools do to this will be most of the IDE’s like IntelliJ or VSCode or even Eclipse. All of which are multiplatform IDE’s. One specific to Mac development is Xcode. | Same with Mac development Linux is fairly straight forward. Since the JVM is available for this platform as well. The same major IDE’s are also available for Linux development excluding the Apple specific Xcode. Since it is mostly free and open source much of our development costs will be low for testing on this platform. | Again the development language will likely be Java. Since we will be able to leverage the JVM. Much of the same IDE’s will be available as well as the power Visual Studio specific to Windows machines. VSCode is recommended due to it being compatible with all three major operating systems. | For Mobile Platforms we will need to use the specific development tools for the Android and iOS environments. Specifically Android Studio using the Kotlin language and Xcode for iOS using the Swift language. While android studio is available for all major OS’s, Xcode can only be used on Mac machines. Another consideration is that we will need developers familiar with creating apps on these platforms. |

## 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**:
   1. I would recommend the Linux platform. Specifically the Ubuntu Server.
   2. It has high scalability and can support the growth of Draw It or Lose It. It is cost-effective being open source and free. Linux is also known for its robust security featurse and regular updates along with its extensive community support.
2. **Operating Systems Architectures**:
   1. The core kernel manages system resources including the CPU, memory, storage, and I/O operations.
   2. It has a robust shell that allows for command line execution.
   3. Hierarchical structure for file storage and organization and supports various file systems.
   4. also supports tools for updating and managing software packages.
3. **Storage Management**:
   1. I recommend LVM (Logical Volume Management) This technique provides improved performance and allows for disk resizing if needed because disks are created virtually out of multiple physical storage disks.
4. **Memory Management**:
   1. Linux uses a variety of techniques to manage memory.
      1. Virtual Memory: Uses both physical RAM and disk space to extend memory availability.
      2. Paging and Swapping: Moves data between RAM and disk to free up memory for active processes
      3. Cache: Efficiently manages data caching to speed up access times
      4. Memory Allocation: Utilizes various algorithms to allocate and deallocate memory efficiently
5. **Distributed Systems and Networks**:
   1. Architecture: Utilize the microservice architecture where each service runs independently and communicates over the network.
   2. Networking: Using RESTful APIs for inter-service communication. and Balance traffic over multiple servers
6. **Security**:
   1. Data Encryption: encrypting data at rest and in transit using secure protocols
   2. Authentication and Authorization: Implement robust user authentication mechanisms and role-based access control
   3. Firewalls and Intrusion Detection Systems: Protect servers from unauthorized access and monitor for suspicious activity
   4. Regular Updates and Patching: Ensure the operating system and all sofware components are regularly updated to mitigate vulnerabilities