

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 | 07/19/23 | Matteo Russo | Initial document write up. |
| 1.1 | 08/02/23 | Matteo Russo | Evaluation filled out. |

**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)

Draw It Or Lose It is an app only available on Android. The goal is to take the app and make it available as a web based game.

The game will have one or more teams involved with each team consisting of multiple players. Game and team names are unique to allow users to check if a name is in use. Only one instance of the game can exist in memory.

To meet these software requirements object oriented design will be employed. Classes will be made to manage smaller classes within them. Game will contain the Teams, and Team will contain the Players. This allows for many teams with many players within them. Each of these will be subclasses of an Entity class that will give them all ID and Name properties. This will allow for management of duplicate names as well as preventing duplicate games.

## Requirements

* The game will run on the web
* A game will have one or more teams
* Each team will have multiple players
* Game and Team names must be unique
* Users may check if a name is in use when choosing a team name
* Only one instance of a game can exist in memory at any given time

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

Given the web-based environment, extra constraints are added to the design of the application.

Programming languages - These are limited as a language that is chosen should be one developed for web-based projects. Not all languages work well in web environments.

Browser support - The game will need to be made to support chosen browsers. This will mainly affect the front end development but may require some back end adjustments.

Hardware - Being web-based it is unknown the specs of the device the game will be played on. A minimum spec should be established for the application to run on as well as deciding on the devices that the application is designed for. Many devices can access the web and play web games such as desktops, laptops, phones, tablents, and even refrigerators.

World wide - Since the application will be available on the internet it will be accessible across the globe. This may result in needed compliance for it to be available within regions such as data tracking laws or internet related acts that may require the app to be more accessible.

## [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)

GameService: This is a class that implements the Singleton pattern, seen by the use of a hidden constructor and stored pointer to an instance of itself. This class encapsulates the functionality for managing existing games and accessing the teams and players. This class contains a list of Game classes allowing it to hold zero to many Game instances.

Game: A subclass of Entity, this class contains a list of Team classes ranging from zero to many.

Team: Another subclass of Entity that holds a list of Player classes ranging from zero to many.

Player: The final subclass of Entity. This class only holds basic information about plates in a team.

Entity: This is a base class allowing for inheritance of its data and functions. It holds the common data of *id* and *name,* along with accessor functions and a constructor, that Game, Team, and Player all share.

**"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** | -High security  -Reliable  -Good workload distribution  -Great support  -Unlimited user license  -Expensive hardware  -Very limited third party applications  -Open source but may need workarounds for MacOS issues | -Free to use  -Runs on cheaper hardware  -Better security  -Open source  -More widely used  -Requires more knowledge about the system for bug fixes, maintenance, and set up | -Runs on medium priced hardware  -Easy to use  -Good third party application support  -Allows use of Microsoft proprietary apps  -Costly OS license  -Vulnerable to threats  -Servers will go down more for updates | -Cheaper hardware if using older smartphones  -Requires less space  -Can not sustain as many connections  -Not as powerful  -May not handle the requirements of the game |
| **Client Side** | Clients will access game through browser. Client OS has no impact.  Popular browsers: Safari, Chrome, Firefox | Clients on desktops will interact with game using keyboard and mouse input.  Popular browsers:  Firefox, Chrome, Brave | Desktop client’s view will be larger resolution and commonly landscape.  Popular browsers:  Chrome, firefox, and Opera | Default device web browser:  - Apple: Safari  - Android: Chrome mobile browser  Touch based input.  Smaller screen resolution.  Portrait aligned view. |
| **Development Tools** | Languages:  HTML, Java, Python, JavaScript  IDE:  -Eclipse: multi-language, free  -NetBeans: free. Supports; Java, JavaScript, HTML, PHP, C++  -WebStorm: $159-$779/year. Supports; JavaScript, TypeScript, HTML, JSON, YAML, Shell scripts | Languages:  HTML, Java, PHP, JavaScript  IDE:  -NetBeans: multi-language support, free  -JDeveloper: free. Supports; java, XML, SQL, HTML, JavaScript, PHP  -WebStorm: $159-$779/year. Supports; JavaScript, TypeScript, HTML, JSON, YAML, Shell scripts | Languages:  HTML, Java, C#, JavaScript  IDE:  -Visual Studio Code: multi language support with plug-ins, $0 - $250/m  -Eclipse: multi-language, free | Languages:  HTML, Java, C#, JavaScript, PHP, Python, Swift, C++, Kotlin  IDE:  All mentioned IDEs are suited for mobile app and web development. |

## 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 or cloud. This will depend on budget, time, and desired amount of control.

Linux is an open source operating software that provides the functionality of an OS, system and data management, but is more bare bones than others. Although lacking in some areas this actually makes it a better option. It is easy to build on which allows for a more intimate knowledge of your own server and has a large source of open source programs that can be easily added to the server without having to develop them yourselves.

While a physical server will provide much more control, a cloud based server is also a good option based on budget. A cloud server may be cheaper and faster to set up as it can be structured around third party software that is already proven to work and allows for easier future expansion as well as easy changes to architecture of the overall server through changes between the third party services.

1. **Operating Systems Architectures**:

Linux will provide an environment that will allow for control over the server such as memory, storage, devices, and its connections. It will have the architecture of a physical server allowing for connected devices to communicate through the web with a RESTful architecture. This of course implies the use of multiple architectural constraints being a uniform interface for interactions, a client-server relationship between the user interface and data storage, stateless requests from the clients to the server, cacheable or non-cacheable responses, a layered system of hierarchical layers for interaction security, and optionally downloadable code from the server to simplify the client side.

Cloud provides a more modular environment that can expand and be changed when needed with less changes needed to the rest of the system. This follows more of a microservice or SOA architecture which leverages different selected services to handle features and functionality for the server, such as security or a database, then tying them together with the host API

1. **Storage Management**:

Linux utilizes a file system that is hierarchy based. This will mean that files have limited access to other files based on their placement in the system which provides an added layer of security but will take more planning on where files should be placed. Of course being a physical server this data will be saved to a disk. Accessing the disk can take time so only important data should be stored there while retaining the rest in memory for faster access and execution.

Cloud storage will be handled by a third party service, however due to the nature of cloud servers no local data may be stored or accessed as any request may be handled by an entirely different machine. This means that requests will need to contain all necessary information when sent.

1. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
2. **Distributed Systems and Networks**:

Communication through networks between different systems is luckily easy as the communication is not platform dependent, as in the OS does not make a difference but the web browser will. Communication is done through https requests that point to different data on a server and these requests are sent via connections to the server from the client. These connections can take many forms such as sockets that define end points for communication, pipes that allow processes to communicate through read and write data flows, or remote procedure calls. In the end the client requests data from a server then interprets it with the information given from the server. This interpretation may be different from platform to platform but will functionally be the same.

To ensure a good experience the client may need a good connection to the server or risk timing out if a request takes too long to respond. In the context of Draw It Or Lose It this may cause players to disconnect from the server in which case a system may be put into place to assist the user if they reconnect. This could be done by keeping the player in the game despite not being connected and temporarily keeping the client ip in memory so that if they reconnect before the game is finished they may resume control without losing all progress in a game and having to connect to a new one.

1. **Security**:

With Linux security is inherently better for the server due to the OS being so customizable. This means because it can be changed so deeply by the server owner, hackers will have less knowledge about the systems in the server opposed to an OS that is widely available and does not allow much change like Windows. Additional steps however should be taken to improve server security such as including the use of encrypted connections to clients to prevent interception of data, setting up a firewall to further control that gets passed to the server, and preventing overflow requests by preventing code execution in a stack section of memory.

For cloud security services may be chosen and added to increase security. Services that create security tokens that provide temporary credentials for users to invoke the API or authenticate a user through sign-up and sign-in functionality to the app through a password or even social identity such as Facebook or Amazon.

For general security for the user however, steps should be taken to protect them such as implementing a secure password check, biometrics, or even one time passwords through two factor authentication. This will also improve server security by only allowing trusted users to access it.