

Draw It or Lose It (Web Application)

# **CS 230 Project Software Design Template**

Version 1.0

## 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)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.2 | <08/16/20> | Danny Faught | Added recommendations section |
| 1.1 | <08/02/20> | Danny Faught | Added evaluation |
| 1.0 | <07/13/20> | Danny Faught | Initial description of web application and design patterns used |

**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 currently runs an Android application known as Draw It or Lose It. They are ready to expand to other platforms and want to start with a web application.

There will be a single game service running always that can house any number of games. Each game can house any number of teams. And each team can house any number of players.

Games, teams, and players must have a unique ID/Name associated with them.

## [Design Constraints](#_2et92p0)

Technical – Since we are coming from an Android application, we are expecting to use the Java language to ensure compatibility with any existing architecture.

Technical – Web applications can run on many different browsers. Multiple environments will be needed to test across all major browsers.

Technical – We will need to ensure that a single service is running all games and that multiple services cannot be created.

## [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)

In this program we demonstrate OOP principles such as inheritance with our Entities, Games, Teams, and Players classes as well as encapsulation in all classes by setting private fields with getters and setters. We also demonstrate polymorphism when we override our print methods or make multiple constructors.

**Entities, Games, Teams, and Players:**

These classes form the objects in which our program will interact with. The Entity class is a base class that is inherited from by the Game, Team, and Player classes. Because each of these subclasses have an id and a name, those fields are with the Entity class. Since we do not want an empty entity to be created, it’s empty constructor is set to private. We have another constructor that creates an entity with an id and a name. Here we also have methods for getting the ID and the name as well as a method that prints those items (this method is overridden by the child classes, however).

The Game and Team classes are both very similar. The Game class contains a list of Teams and the Team class contains a list of Players. There are constructors for each that take an ID and a name and call the parent class’s (Entity) method to construct. The Game and Team classes also have a method to add a Team or a Player to its list respectively. These methods take a string and check the list for a team/player of the same name. If it doesn’t exist, the team/player is added to the list. Here we also override our print method to print the number of teams/players as well as the ID and name from the parent class.

The Player class has a constructor that creates a player with an ID and a name and a second constructor to add a player to a specific team. The second one takes a team as an argument. This class also overrides our print method to print the player details.

**GameService:**

The GameService method serves as an environment for our games to take place in. The GameService can hold any number of games. This environment is written with a Singleton pattern to ensure there is only one instance of our environment running on the machine. We do this by making it’s constructor private, creating an instance of our service, and then use a getter method for the instance when we need it.

This class contains a list of all the games being played. It also contains the game, team, and player IDs which are incremented each time a new instance of any one is created.

We have a method here to add a game which takes a name and searches the list for an existing game of that name. If it finds one, it returns that game. If it doesn’t, it adds that game to the list.

There are also two methods to search for an existing game. A game can be searched for by either a name or by its ID.

A method to get the number of games exists. Also methods to get the player or team ID fields. These methods also increment the field.

**ProgramDriver:**

This is our main method. This is where we initialize our game. In our example we create several games, teams, and players. We also use SingletonTester here to test our design pattern.

**SingletonTester:**

To ensure our Singleton pattern is working as intended, we call the instance from a separate class and count the number of games. It should return the number that were created in ProgramDriver if functioning correctly.

****

## [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** | Mac is also a stable operating system as it is based on the BSD architecture.  Mac is also easy to use as it implements a GUI that a user can choose to use.  Mac is not as common, so some platform-specific technologies may not be available.  Could be pricey to scale to large number of clients | Linux distributions are usually free. They have proliferated the web-hosting market making them widely available and cheap in comparison to other options.  Linux is known to be stable and require little rebooting, even after changing configurations.  Linux typically has better performance than other types of operating systems because of their typical “barebones” architecture.  Several solutions work out of the box with Linux web-hosting. Many tools have been developed in the Linux web-hosting space with great support communities.  Linux can be more difficult to use for users who are used to GUI based platforms like Windows. | Unstable  Easy to use  Windows specific technologies  Since the Windows platform is the most used OS in existence, many users may find Windows easier to use than other options.  Also, because of this reason, many Windows-specific technologies exist that may be needed for a web server.  Windows is known for being unstable (blue screens, frequent restarts), so it may not be the right choice for a server needing 100% uptime.    Could be expensive to scale to large number of clients. | Mobile devices have the benefit of being cheap and portable.  They are not powerful enough to scale to accommodate large amounts of users like the other platforms.  Mobile devices are power inefficient for this type of computing.  Very little support for serving from mobile devices exist and custom solutions would likely be required. |
| **Client Side** | Mac uses Safari web browser by default, so testing will need to be to take this into consideration. | There are many different distros of Linux testing will need to consider at least the most common ones. This will take extra time and cost a little more than other platforms.  Developers will need to be comfortable in Linux | All browser types will need to be considered for Windows as users tend to be pretty spread out here. | Responsive web design will be a must as screen size will need to be a consideration here.  The client application should be small to accommodate for use over mobile data  There are many types of mobile devices, especially in the Android realm. This will require extensive testing on various devices or emulated devices.  Developers will need to be comfortable with both Android and iOS. |
| **Development Tools** | VS Code  Git to collaborate with other devs | VS Code  SSH tool to access server remotely  FTP tool to transfer files from other platforms.  Git to collaborate with other devs | VS Code  Git to collaborate with other devs | Android Studio  iOS SDK  Git to collaborate with other devs |

## 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**:

For a service that allows Draw It or Lose It to expand to other computing environments, Linux would be the best operating platform to utilize.

1. **Operating Systems Architectures**:

When comparing Linux to other architectures, we find that it is preferable for many reasons. The biggest of which is performance. Linux distributions typically outperforms other server architectures by a large margin. This can mainly be credited to the stripped-down nature of Linux. There will not be a GUI utilized in our chosen version of Linux. This means no computing power will need to be allocated to visual elements such as windows or animations. This means our developers will need to be versed in Linux navigation and scripting, but Linux is easy to pick up and will be well worth the effort for the performance gain over other platforms.

Another reason we chose Linux is that it is very well established as a server platform. This means that we will have ample resources available to us when developing a web server. The other platforms available to us can all be used to serve web components, but the number of resources available in which to do so pales in comparison to the proliferated Linux.

We could also go serverless with this game (with AWS or other services) but given the scale we are currently operating at and the cost-benefit of going serverless (storage and database call costs), choosing to manage our own server is preferable at this time.

1. **Storage Management**:

With our game comes the need for unique user profiles, storage for 200+ high definition images, as well as implementation of the libraries used for securing our application.

For the user profiles, the accepted way to handle this is by using a secured database that can effectively store each profile with a unique identifier.

Our images can be stored in a typical hierarchical file structure. The same can be said for the libraries used in our application.

For all storage aspects of our application, we will have built-in redundancy in the case of crashes, data loss, or outages. This can be in the form of extra copies of our application and file structure on separate media that can be easily switched to when needed.

1. **Memory Management**:

We intend on our server being a high-performing machine that can serve many users at one time. This means that the amount of memory our system requires can be high. When choosing a machine, we will want to build something that both has enough memory to handle our best-case scenario for number of users (a lot!) as well as being expandable in order to grow later when needed or in the case our game exceeds expectations. We will also have swap storage implemented on a part of our storage medium for added flexibility.

When writing the application, we will also want to consider events that force garbage collection or make them last longer than normal. Garbage collection is automatically handled in the language that we have chosen when an object is no longer needed. However, if the application is written in such a way that garbage collection is called multiple times in the middle of an event when it would be more efficient to call it once at the end of the event, this can degrade the performance of our application.

1. **Distributed Systems and Networks**:

For our application, since it is web-based, we are using a client-server pattern to server our game to clients on various platforms. This means that any platform that has internet access and can use HTTP requests in a browser could potentially use our game!

On our side, we will also need to ensure that we have a dedicated, high-speed internet connection. This creates high performing interactions with our server since no other services will be using the internet connection. It would also be wise to implement a back up network in the case of a system outage.

1. **Security**:

For users of the system, we will need to ensure that each has access to only the resources needed to interact with the necessary parts of the game. For example, a player would not need access to the administration resources of Draw It or Lose It. This can be set up using a role-based access control system, a well-documented pattern in which roles are assigned only the privileges needed.

For our database of user profiles, we will need to ensure that user information is properly encrypted in the instance of an attack by a bad actor. This is done by libraries we have included with our application.