

Draw It of Lose It

# **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 |
| --- | --- | --- | --- |
| 1.0 | 11/14/21 | Brian Golden | Added Executive summary, design constraints, explained the domain model, updated the evaluation table |
| 2.0 | 11/25/21 | Brian Golden | Filled out table with server and client side requirements as well as the development tools. |
| 3.0 | 12/7/21 | Brian Golden | Filled out the overall requirements for the project. This completes this document. |

**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)

We need to create a web-based game that has the ability to have one or more teams assigned to a single game. Each team will have multiple players. We need to check to make sure each team name is unique for each game. Only one instance of the game is allowed at any point. We can use the singleton design to ensure that only one game exists at any point. Teams can be assigned to this game, quickly checking the name of each team before they are added to make sure there are no duplicates. Players can then be added to each team.

## [Design Constraints](#_2et92p0)

The biggest constraint of this project is that in a web-based environment there can be a lot of people accessing this at any point. We need to make sure that, at least at this point, we lock down the ability to join a game once it is started. Since only one game is allowed to be running at one time, we need to make sure that that game is not interrupted once it is started.

## [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 GameService object allows a zero to many relationship to the Game object, meaning that the GameService can run no games or a lot of games. The Game object has a zero to many relationship with the Team object, meaning that there can be zero Teams assigned to a game or a lot of teams assigned to the Game object. There is a zero to many relationship between the Team and Player objects as well. This means there can be zero Players assigned to a Team object or a lot of Players assigned to a Team. The Game, Team, and Player objects all inherit from the Entity object.

**"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** | Using a Mac for the server is not ideal. Mac’s have a very high cost for the computers themselves. It is also not easy or cheap to add more storage or computing power. | Linux would work very well. The operating systems are free and can run on most hardware. They are also lightweight on storage. This might require more technical experience as the command line will need to be used extensively. The OS setup itself can also take some work to get set up. | Windows can easily be used to set up a webserver. There are special versions of windows that are designed for servers. There are also a lot of applications that work easily on windows. Hardware is easy to upgrade and expand as well. | This is not recommended at all. Mobile devices have low storage and low computing power. They are also generally not able to get connected to a hardwired internet, which creates a low bandwidth. |
| **Client Side** | There is not an issue using client-side software, especially on a web-based program. Macs have support for many different browsers. | Using a web-based platform with linux clients is not hard to set up as well. Linux has support for chrome based browsers which we would need to develop anyways, as Chrome is one of the most widely used internet browsers across all operating systems. | This will not require any extra work with our web-based approach. There are many browsers that windows supports. | This will require a little more support. Since mobile devices run at a smaller aspect ratio and have less computing power than computers, the web-based platform will need to be optimized for mobile devices. |
| **Development Tools** | To even use a server on a Mac, there is a $20 program. The server itself is an Apache server which supports many major scripting languages so there is not any special software needed to make the server software. Since this is web-based, the client-side system would use a combination of HTML and CSS which does not need any special software to create. | To run and develop a Linux based system, there is not any up front licensing costs. This can be done any most hardware and just requires a simple download. Any server based software is also made using a combination of Apache, MY-SQL, and PHP, all of which do not need any licensed software to use. The client side also just uses a combo of HTML and CSS so that does not require special licensed software either. | To actually run a windows server this can cost anywhere from $500 to $6000 depending on the amount of servers you have and what you want them to do. This does not include the hardware that they run on either. After that there is not any special software for the client or server based software that needs any special licensing or cost. | Running the server, on an android at least, just needs a little free app. This is very basic but allows you to share files with a computer so it does not cost anything. Since the abilities are so limited there is no need for special software to develop the server software. The client software would still just use a combination of HTML and CSS which does not need any special software to make. |

## 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**: The server would be best run on a windows server. This allows the maximum amount of 3rd party support, as well as a relatively cheap startup cost of buying the software. This will allow us to easily expand the hardware of our server if need be.
2. **Operating Systems Architectures**: When using a Windows server to run our game, we have access to a large amount of 3rd party software which will allow us to expand our capabilities as needed. Windows also runs on any hardware, which will allow us to expand and upgrade our server as needed. A windows server will service any other operating system so this does not tie our project to a Windows only environment. There are also many support options to run a Windows server, between Microsoft’s support team and many online resources.
3. **Storage Management**: For the application storage, i.e., code and some assets, we are going to hold that on the hard drive of the server itself. For all 200 images we are going to be using, we will set up an AWS S3 bucket to hold all those images. Since there are so many high-resolution images, having them on a cloud platform allows us to save storage memory on our own server as well as save bandwidth on our server, calling the AWS S3 bucket to download images directly to a user’s browser.
4. **Memory Management**: The biggest RAM saving aspect of this project will be storing and retrieving the images from a cloud storage solution. Our program will download a couple images ahead of time, only storing up to 3 or 4 at a time, so that way the program does not load 1.6 gigabytes of images alone into RAM. Having a few downloaded at a time allows them to still display on demand like they should but keeps the overall memory usage low.
5. **Distributed Systems and Networks**: As this project expands in popularity, it might be a good idea to move the entire project into the cloud. Instead of running a windows server directly on hardware that our company must manage, power, and maintain this entire program can be moved on to a cloud service like an AWS EC2 instance. Using this cloud-based approach allows the program to run all the time on a platform that is fault tolerant and can run on multiple different regions across the world. This allows the program to run anywhere in the world and is readily available to anyone that wants to use it. Using an AWS EC2 instance allows our company to take the entire existing project and just move it into a windows server that runs on that instance. If we need to expand the project to service more people at once, multiple copies of the same EC2 instance can be created instantly, allowing the game to always run without any issues.
6. **Security**: For this game there is little to no user data stored. Each game has team names and usernames that are stored during an active game, but that data is not stored long term on our servers. Since the data is stored temporarily, it will be encrypted and isolated to the single game that each team and user is part of. Any images that are sent between the server and clients will also be encrypted and only decrypted as they are displayed into the client’s screens. Also only authorized users will be able to access team names and usernames. These accounts are password protected and only available via very specific REST API calls. This allows the minimum amount of access necessary to make the game run as it should.