

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 7/21/2024 | Linkhon Hasan | The Gaming Room has an Andriod app, and they would like a web version of their game app. |

**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 has an Android version of their game, Draw It or Lose It, but they lack the knowledge to replicate their Android app as a web application. In order to have a web version of this game, a front-end is necessary, which users can interact with and play the game. Also, since the application will use stock drawings, we would need a database to hold all the images and store player names, team names and scores. A server that serves the data from the back end to the users would be a good fit for this app. We should also implement a buffer which loads the image chunk by chunk, so it takes 30 seconds for the image to fully load.

## Requirements

The client would like the game to be played by teams; each game session should have one or more teams playing them, if more than one team then they would be competing against each other. The client requires multiple players per team and a player can only be in one game at a time. Also, the client requires all team names and game names to be not similar, so if a game or team name “X” already exists, another game/team cannot be of the same name. So, the requirements are unique names for teams and games, individual list structures to hold multiple games, teams and players, and checking implementation so we can find if a game or team with the incoming name from users already exist in the system.

## [Design Constraints](#_2et92p0)

We should set a maximum team limit per game, and if there is only one team in a game, players can just play without competition against anything. Each team should also have a limit on how many players can be in it. When creating Game and Team, checks must be performed to make sure the names do not already exit in the backend. So to save memory, the names must be deleted once an instance of a game ends so those names become available again. We should have unique ID for every player and teams that is tied to a game that they are in, so this way if their id tries to start another game without exiting the current game, their request will not work. Since this will be a web-based game, we should ensure the client side is compatible with all available browsers across all devices. As for the backend, it will not affect how the client side of the game is being distributed.

## [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 UML Diagram provided by The Gaming Room shows an abstract Entity which holds a name and an id, and its fields and methods can be inherited by three other classes. This is known as inheritance in object-oriented programming, and the first class is a Game class which inherits everything from Entity and can hold a list of zero or more instances of the Team class. The Team class also inherits everything from Entity and can hold a list of zero or more instances of the Player class. Finally, the Player class also inherits everything from entity. All four of these classes have their own toString method that allows us to print their information as a string. The Game service class can hold a list of zero or more instances of the Game class and has member fields that hold the gameId, playerId and teamId. This will be useful to perform unique checks, and has a field named service that is an instance of itself. It also has an empty private constructor and a public method to get the instance from the fields. This suggests that the GameService follows a singleton pattern. Other member methods in the GameService indlude adding a game to its list, getting a game, team or player. It also has a method to get the total number of games in a session. Finally, the ProgramDriver is the driver class which uses the SingletonTester class for testing.

**"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** | Mac is UNIX based and offers very high performance. UNIX is similar to many prod environments and developers familiar with iOS will have an easier time working on the app. Will make porting to Apple devices easier. Potential licensing costs along with the expensive hardware will require a bigger budget. | Also Unix based and will have lower cost as Linux is free. It is highly customizable and can be configured to our needs, making it ideal if security is a big concern. But software compatibility can be a challenge and developers will need to be comfortable with Linux. Will not affect the budget greatly due to the open-source nature. | Very user friendly and has lots of tools available for developers. Compatibility is usually not an issue, but UNIX based tools or codes will need third party tools causing performance to take a hit. Depending on the type of Windows license we use, could affect the budget. | Server side will not need a separate back end for mobile devices as our app will be distributed via web. Hence, all mobile devices should be able to access our game via web browser without needing to host any server-side codes. |
| **Client Side** | Mac has great support for front end building tools, and all frameworks are supported. All modern IDE tools are compatible with mac, but the challenge may be providing mac pcs to developers as they are rather expensive. This along with licensing costs will require a bigger budget. | Client-side development is possible in Linux but compatibility and learning curve is a concern. Although it can be comparably more secure, and is free to use, developers familiar with Linux will be more costly. | Has excellent compatibility with development tools, frameworks and IDEs. Most developers are very familiar with windows devices and easy to find support for trouble shooting and tools setup online. Not as efficient or performant as mac but cheaper and less expensive in comparison. | Should not be used to development, but our front end must be responsive and work well on screens of any size. Since a separate codebase is not necessary for mobile devices, it will not affect the budget. |
| **Development Tools** | Very good compatibility with dev tools, but some tools may require additional tinkering to get working on Mac. Licensing cost for tools should be the same as other platforms but combined with the higher hardware costs, budget will need to be increased by a large margin. | Limited compatibility for tools, it is well suited for something custom due to the highly customizable nature of Linux, but for our purposes, the customizable feature is not needed. Most dev tools are free and open-source, but will need more experienced developers within the team which will lead to a much higher development cost. | Very good compatibility supports all the tools needed for our application natively just like Mac. Easier to set up most dev environment and offers more free to use software and tools for developers. Cost will be lower than Mac but tools used may not perform as well as their Mac counterpart. Developer team cost should be similar to mac. | We do not need any mobile specific tools as we are not making native mobile versions but web versions that mobile devices will be able to access. The client should follow reactive front end design and it incur any additional costs in terms of dev team as front end developers are familiar with building responsive websites. |

## 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 recommend macOS due to its high performance, stability and lots of development tools supported by the platform. Also testing our game app for the Safari browser will be much easier when developing in MacOS.
2. **Operating Systems Architectures**: macOS has a Unix-based architecture which is combined with Apple’s proprietary additions making it very efficient when it comes to multitasking and memory management. macOS has a XNU kernel, which is a Mach microkernel combined with some specific features from FreeBSD, making it highly scalable and modular. This makes it very well suited for hosting our game server. Although macOS supports both x86 and ARM CPU architecture, the arm64 is preferred as it offers better performance at a lower power consumption rate.
3. **Storage Management**: SSD are recommended for our storage solution and macOS features Apple File System which is optimized to make full use of SSD and can work with large files at a very high transfer rate. This is ideal as our back end will be accessing and using a large number of images from the storage device. APFS also offers features such as snapshots, which is very useful for data recovery, and native encryption of data, which enhances the security of the back-end data.
4. **Memory Management**: macOS has an excellent memory management solution where it makes use of dynamic paging. Essentially, it stores the data that is accessed and used by applications very frequently in RAM, which drastically speeds up the loading of game assets and images for our game. It also supports virtual memory, which uses disk space when RAM is full, this ensures that the system does not freeze or crash under heavy load, making our application and the hosted platform much more stable and reliable.
5. **Distributed Systems and Networks**: macOS supports the development of applications that use RESTful APIs; we can use it in our backend to support compatibility and distribution of the data needed by clients running on different platforms. We can also integrate iCloud support in the server side so in case of an outage, the clients can make use of the cloud service to gather necessary data. macOS also uses WebSockets, which is a single bidirectional connection with a long-lifespan unlike HTTPS, this will help in faster data transmission between the server and multiple clients without a large overhead of repeated request-response-based connection.
6. **Security**: macOS comes with a variety of built in security features that we can leverage for our application. Gatekeeper is a great aspect of the OS ensuring that applications and tools that are verified and tested to be safe can be installed in the system, this will greatly reduce the risk of malware or ransomware attacks on our system. FileVault is a macOS native security measure where the entire storage disk is encrypted. This combined with our in-app data hashing layer will provide a well-functioning secure storage for sensitive user data. macOS also uses the Unix permission system, where system critical resources are restricted from being accessed. To further protect the system and data integrity, this platform also offers a native antivirus, XProtect, which provides continuous security against known threats.