

Draw It or Lose It (Web-Based)

# **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 | 07/25/2024 | Jordan Santiago | Initial document creation |

**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)

Our client, The Gaming Room, has a game called “Use It or Lose It” which is currently available on Android. However, they want to develop a similar game that is made available as a web-based game, serving multiple platforms. The Gaming Room does not know how to set up the environment and has contracted Creative Technology Solutions to present a solution.

The problem that must be addressed is the creation of an environment where multiple people, on multiple teams, are able to play in unique game instances. To that end, all games and teams will need to have unique identifiers, and the system will need to be able to check and report on if a team or game name already exists. The software we will create will include a game management class, which will manage the single instance of the game and the storage of unique identifiers. It will have functionality to add new games, teams, and players. Lastly, it will be able to run the game from beginning to end.

## Requirements

1. A game will have the ability to have one or more teams involved.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
4. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.
5. The game must be web-based and playable on multiple platforms
6. The game must be compatible with various web browsers

## [Design Constraints](#_2et92p0)

1. There may be a budgetary constraint, which could limit the number of resources available for developing this application.
2. There could be a time constraint, which could limit the depth of the development, given the number of resources allowed.
3. The developer needs to be familiar with applications across multiple platforms and web browsers in order to successfully build the application and have it function for all the intended technology.
4. The server hosting the application needs to be strong enough to support a large number of players without a degradation in performance.

## [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 ProgramDriver class will contain the main method, which is where the body of the game application will be contained. The ProgramDriver class has a direct link to the SingletonTester class, which is where a test for the singleton included in the GameService class is located. The SingletonTester class will be removed prior to being released for production. The Entity class will act as a parent class for the Game, Team, and Player classes, with have an inheritance relationship with it. The Entity class will hold common attributes and the three children classes will inherit those attributes from it. The GameService class will contain the single GameService instance and manage it to make sure that there is only ever one in the memory at a time. Additionally, it will contain the functionality to create a Game object and add it to a list of games. The Game class contains the functionality to create a Team object and add it to a list of teams that are contained within a specific Game object. Similarly, the Team class contains the functionality to create a Player object and add it to a list of players, which are contained within a specific Team object. Lastly, the Player class defines what a Player object is.

There are several object-oriented programming principles demonstrated within this UML diagram. First, there is the inheritance that was previously described. The Game, Team, and Player classes will all inherit their attributes from the Entity class. There is also encapsulation, which is demonstrated by having the code separated into different classes and their attributes locked, limiting direct access from unauthorized people. There is also polymorphism, which is demonstrated in the GameService class by the two getGame() methods. There are two getGame() methods that are used to return data of a different type. Lastly, there is a 0 to many relationship between many of the classes. The Player class has a one way 0 to many relationship to the Team class. The Team class has a one way 0 to many relationship to the Game class. The Game class has a one way 0 to many relationship to the GameService class. This means that the one GameService may have many games, which may have many teams, which may have many players. However, a player cannot have many teams, teams may not have many games, and games may not have many GameServices.

**"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** | Has built in server hosting capabilities. However, licensing can be costly, and development would need to be done on a Mac. | Ubuntu is a stable, secure, and easy to use Linux distribution for hosting a web server. The cost is relatively inexpensive. | Windows is user friendly and stable. However, it is less secure that hosting on Linux and more expensive. | Mobile devices lack the power to host a complex web server, and certainly lack the potential to scale. |
| **Client Side** | Mac has a great support network for development. There are a lot of developers with expertise in development on Macs, so the time needed will be less. This will likely cost less than Linux and a similar amount to Windows. | Linux has a lot of options and flexibility, which makes expertise a large factor in client-side development. There is relatively little formal support for Linux development, which can increase the amount of cost and time needed in development. | Windows has a great support network for development. There are a lot of developers with expertise in development on Windows, so the time needed will be less. This will likely cost less than Linux and a similar amount to Mac. | Mobile devices can refer to a range of operating systems that would need to be considered to fully grasp the time, cost, and expertise requirements. Ultimately, a developer would need to be versed in various technologies and programming languages to be successful, suggesting that the requirements will be higher for cost, time, and expertise. |
| **Development Tools** | Macs can support development in many programming languages, including Python, Ruby, Swift, and more. Some common IDEs that can be used are Visual Studio Code, and PyCharm. | Linux can support development in a plethora of programming languages. Some common IDEs that can be used are Eclipse, Visual Studio Code, and PyCharm. | Windows can support development in many programming languages, including JavaScript, Python, and React. Some common IDEs that can be used are Eclipse, Visual Studio Codem, and PyCharm. | Mobile devices can support development using Kotlin, JavaScript, and Python. Depending on the type of device chosen, a couple of common IDEs are Android Studio and Cordova. |

## 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**: Considering all the factors, I would recommend completing the development using Windows. Windows is more flexible than Mac but has more support than Linux. In terms of cost, time, and expertise, Windows is similar or better than the other options.
2. **Operating Systems Architectures**: Windows has a layered architecture, with the two main components being user mode and kernel mode. It is multifunctional and capable of multitasking.
3. **Storage Management**: Amazon FSx can provide a cloud storage solution for the game that works well with the Windows file server. This is a relatively low-cost option that would support scalability and frequent updates. It will allow the developer to store any images needed for the game in a place that can be easily accessed by the application. Additionally, as the application is expanded, a cloud service will allow additional storage to be purchased to make room for additional images.
4. **Memory Management**: Memory for Windows is managed dynamically, with it being allocated and deallocated between the application’s local memory. The memory would be allocated for the duration of the game and then deallocated at the end of the instance and contain any data necessary for running the game instance. Therefore, if a game instance needs ten images, the application will pull only those ten images out of the hundreds in storage to use during the game. The methodology supports superior game performance with a tradeoff of
5. **Distributed Systems and Networks**: Cross-platform communication and functionality can be easily accomplished by using a framework that already can be utilized across various platforms. For this I would recommend using React Native to develop the software, which can function with Windows, Linux, Mac (somewhat limited), and various mobile devices. Using a client-server distributed system and the internet as our network, we can have multiple computers accessing the game, at the same time, from multiple locations.
6. **Security**: The best way I think we can protect the security of the users is to use secure coding development in creating the software. This will include encrypting user data and providing authentication layers for validating if someone has access to the data. Additionally, using a Virtual Private Network (VPN) can help encrypt user data and mask their hardware data.