

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 06/09/2024 | Dillon Merriam | Executive Summary and Design Constraints |
| 2.0 | 06/14/2024 | Dillon Merriam | Evaluation Updated |
| 3.0 | 06/23/2024 | Dillon Merriam | Recommendations Updated |

## [Executive Summary](#_sbfa50wo7nsh)

A web-based version of the Draw It or Lose It Android app is to be developed. The environment needs to be created for this game, which will be accomplished in a variety of ways. To ensure the uniqueness of game and team names, all existing names will be iterated over when creating a new name to check if a given name already exists, and only created if the name is not in use. To ensure that only one instance of the game exists in memory at a given time, every game, team, and player will be unique.

## Requirements

The Gaming Room needs to set up a web-based environment for their app. The game needs to have multiple unique games and teams, of which consist of multiple participants.

## [Design Constraints](#_2et92p0)

* Each game will have one or more teams. This means there will be an array of teams in a game.
* Each team will have multiple members, meaning there will be an array of members in a team.
* Game and team names must be unique, meaning they must be iterated over to check if a name is in use.
* Only one instance of the game can exist in memory at a given time, which will be accomplished by giving each game, team, and player unique identifiers at creation.

## [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 UML diagram, the properties of Class GameService applies to zero or more instances of Class Game, which apply their properties to zero or more instances of Class Team, which apply their properties to zero or more instances of Class Player. Class Entity inherits the properties of Classes Game, Team, and Player. Class ProgramDriver uses SingletonTester to test if a Class (GameService in this case) is of singleton design. The methods of GameService are accessible from anywhere in the program, and allows for the creation of games as well as the access of data of a game. Each Game has Teams, and each Team has Players – all of which are an Entity.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | More expensive proprietary hardware with built-in software that does not particularly cater towards hosting a server, but it certainly can be done. | Lightweight OS that is a popular choice for server hosting, with a wide range of distributions to choose from to cater towards the user’s needs. Linux is free and open-source, and because it is especially customizable, it can be very secure. | The most widely used OS in the world that is capable of flexibly running many different programs, and emulating operating systems if needbe. | Mobile devices are not as powerful as desktop systems, connect to the internet wirelessly which is less reliable than a wired connection. The only benefit is that they are mobile, so they could serve as portable servers. |
| **Client Side** | Macs are known for being easier to use, but the most expensive out these 4 different platforms. Mac uses the Unix system, similar to Linux, with a terminal that I find more intuitive than say Windows PowerShell. | Often known to be the hardest operating system to use for consumers but the cheapest (being free). Time may need to be spent learning the file system, but when you do, it is one of the best operating systems to develop on, giving the developer a lot of control and power. | Windows allows the use for the most amount of tools and software at a modest price and is easy for just about anyone to learn. While all other operating systems have more obstacles in supporting multiple types of clients, Windows has more tools to be able to handle these since it does not rely on as much proprietary software or limited by open-source software. | Mobile devices are by far the simplest and easiest to use, because they are designed that way. This comes as a drawback that they are the worst for developing, or doing any serious work on. Websites need to specifically have mobile versions for them to render correctly on the devices, and it would be near impossible to do any serious work or programming on a mobile device. |
| **Development Tools** | Apple’s free IDE Xcode may be used, as well as Visual Studio Code (which is available on Windows and Linux as well). NodeJS may be used for backend. | The Bash terminal is a powerful tool in the Linux arsenal, capable of scripts and easy maneuvering of the file system. Popular IDE’s include Eclipse, PyCharm, and of course, VS Code. There is a wide variety of open source software such as Vim (text editor), and it can be used in the LAMP stack (Linux, Apache, MySQL, PHP). | Often uses C# and the .NET framework as well as Visual Studio for an IDE. As mentioned before, it is a flexible OS, and tools such as Docker and WSL (Windows Subsystem for Linux) can allow developers to use a variety of tools how they see fit. | Java if working with Android, and Swift if working with iOS. There are ways of developing apps that can allow developers to essentially write for both at the same time using tools such as React Native (JavaScript) however. |

## Each of the above platforms offers a server-based deployment method. Hosting a server on a mobile device, while technically possible, would be the hardest to implement and very unnecessary for our requirements. The other 3 operating platforms are all feasible, though Linux is probably the best choice for its cheapest cost, best security, and excellent development environment.

## Choice of server hosting options will be affected by the demand of processing power, memory, bandwidth, and other factors which will also affect price. Ultimately, server licensing costs will be cheaper using a Linux platform as opposed to Windows or Mac since the latter are proprietary while Linux is open-source.

To ensure that the application will be capable of running on all browsers, it should be tested on a variety of the most popular browsers and written in code that utilizes libraries and frameworks that are compatible with a wide range of browsers. HTML, CSS, and JavaScript should be written with best practices.

Visual Studio Code is free for commercial use and can run on Windows, Linux, and MacOS. At least two development teams (one for front-end and one for back-end) should be involved to design the application and development environment (front-end), and to set up the database and client-server pattern using perhaps MySQL and NodeJS. The MySQL licensing cost varies depending on our needs, and could potentially cost thousands of dollars.

## 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 that the application be built within a Linux development environment as well as hosted on Linux servers. This is because of the advantages it provides in cost, security, favorable development environment, as well as being the most lightweight OS between Windows and MacOS. It is a very common platform for servers, which has helped produce a lot of support and community resources for it.
2. **Operating Systems Architectures**: Web-based applications are rendered through HTML and CSS to display the content and through JavaScript to allow for reactivity to user input. The server-side can be implemented in a variety of ways, through databases constructed by MySQL, MongoDB, Firebase, or other options alongside a tool like NodeJS. From personal experience, I would suggest the MERN stack (MongoDB, Express, React, Node), as it keeps most of the application written in JavaScript which is a powerful high-level language.
3. **Storage Management**: Data that is preserved between games and holds the logic for the functioning of the application should be stored server side and interacted with through a RESTful api to handle requests and responses. User profiles that holds data such as account specific name, password, wins/losses, payment info, etc should be encrypted and stored on the server. The application itself should be as well.
4. **Memory Management**: Individual game state should be created, read, updated, and deleted as necessary. Data that is more temporary and subject to change should be cached client-side. JavaScript has methods of “garbage collecting” to deallocate memory that is deemed to be of no further use by certain algorithms.

Client-side storage can come in the form of a variety of implementations such as localStorage which persists over browser sessions, sessionStorage which exists while the page is open, and cookies which can allow a website to recognize a user and remember their data from past visits to the website among other things.

1. **Distributed Systems and Networks**: The game will run on a centralized server or cloud infrastructure that shares a single database with information that is compatible with all platforms. An error handling system will be implemented to pause the game in the even that there is a loss of connection, and a RAID system will be implemented to back up data in the event of crashes or outages.
2. **Security**: User profiles may be implemented with a username and password which are hash encrypted to prevent the exposure of user data in the event of any malicious hacking. Methods such as Secure Shell Protocol may be used to maintain the privacy of communication between users.

Pen testing will be conducted, and modern cybersecurity protocols will be implemented to ensure the security of sensitive data, as well as the integrity of the application.