

Draw It or Lose It

# **CS 230 Project Software Design**

Version 3.0

## 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 | <05/20/21> | Jacob Valdiviez | First initial version |
| 2.0 | <06/04/21> | Jacob  Valdiviez | Second version of software design |
| 3.0 | <06/19/21> | Jacob  Valdiviez | Third version of software design |

**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 wants to create a game called Draw It or Lose It. Draw It or Lose It is a similar game to the Win, Lose or Draw game. It is a web-based game that serves multiple platforms. This game will have a web-based interface that can allow the game to communicate between multiple platforms. It will only have one instance of the game so it can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

The software requirements of Draw It or Lose It are a game will have the ability to have one or more teams, each team will have multiple players, game and team names bust be unique, and there shall only be one instance of the game in memory. Another design constraint is that there should be a web-based interface to allow communication between multiple platforms. The programming languages that will be used is Java for the client side, Php for the server side, and JSON for the interface. The server must be run on a web host with any preferred server operating system i.e., Linux. Since the game can be run on multiple platforms, it should have the ability to run on Mac, Windows, Linux, and any mobile operating systems i.e., Android and iOS.

## [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 class structure in the client side of the application is based on a singleton approach. The GameService class is the singleton. Any class, including the ProgramDriver, can get the current instance of the GameService class. This class has a list of Game objects. It also holds the current ID number of each game, team, and player. This allows unique identifiers for each class. The Game class has a list of Team objects. It can add a unique team. It will check if the team’s name already exists in the list. The Team class has a list of Player objects. It can add a unique player. It will check if the player’s name already exists in the list of players. The Game, Team and Player objects are children of the Entity object. The Entity class holds basic information of each entity. It holds the unique identifier, and the name of the entity. You can also retrieve the id and name from the class. The ProgramDriver is what runs the GameService. The ProgramDriver also uses the SingletonTester to test if the singleton is working properly. The structure of the server side should follow a client-server architecture. The client of the application will send a HTTP request that will use an API that is integrated with authentication. If accepted by the authentication, the request will be sent to the server. The server will use a database language in a form of a query to retrieve the information that is requested from the database. The server will respond back to the client in the form of a JSON syntax.

****

## [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 must work together.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Hosting on Mac is a closed platform. Mac can release updates that include tweaks. Every time a new version is released you must pay for the new version. Mac hardware will be required to run. Mac web hosting is the most secure. Mac hosts run Apache servers like Linux. | Hosting on Linux is an open platform. Linux web hosting is the cheapest of the three types. The amount of hardware resources needed to run Linux is often less than Windows and Mac. Linux is open-source and allows custom security to be added. | Hosting on Windows is a closed platform. Windows often releases updates. You must pay for a new version of the Windows server. Windows usually goes through updates often leaving your server occasionally down for updates. Windows servers can run ASP, .NET, Microsoft Access or MSSQL databases. | Mobile Devices lack the capability for hosting a web-based application. |
| **Client Side** | A Mac system will be needed to develop the mac application. Expertise in Swift programming language is needed. Browsers supported on a mac are Chrome, Safari and Firefox. | There are multiple Linux Distributions. Need expertise in C, C++, Gtk and other various programming languages because there are multiple different Linux distributions. Browsers supported on a Linux is Firefox, Chrome, Edge and a distribution specific browser. | Any system can be used to develop a Windows application. Visual Studio and C++ expertise is needed to develop a windows application. Browsers supported on a Windows are Chrome, Firefox, and Edge. | A Mac system will be needed to develop iOS mobile applications. Any system can develop Android mobile applications. Expertise in Swift and Xcode is needed for iOS. Expertise in Android Studio and Java or Kotlin is needed for Android. Mobile devices don’t have the capability of developing applications. Browser supported on iOS is Safari. Browser supported on Android is Chrome. |
| **Development Tools** | Xcode and a mac system is needed to develop mac applications. The programming language used is Swift. A browser or the App store is needed to download the tools. | There are multiple IDEs for developing applications for the multiple Linux distributions i.e., Gnome Builder. A Browser is needed to download the tools onto the system. | Vistual Studio is needed to develop a Windows application. C++ language is used. A browser is needed to download the tools. | Android Studio is needed to develop an android application. Xcode is needed to develop an iOS application. Java or Kotlin is used for Android. Swift is used for iOS. There is no support for developing applications on a mobile device. |

## 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 operating system I recommend is MacOS. This operating system will allow the development of iOS, Mac, Android, Windows and Linux applications. The server side I recommend is Linux. It is cheap and reliable compared to the other two. However, Linux servers are not very secure. Server side security, including basic authentication and hash encryption, is needed to protect the server from attacks.
2. **Operating Systems Architectures**: MacOS has the capability of developing iOS and Mac applications. MacOS is needed, so that The Gaming Room can expand its software to Apple products, instead of being confined to just Windows and Android users. For the server side, Linux does not go through many updates. So, its uptime is very reliable. Linux does not require high-end hardware to run. So, it is cheap because high-end hardware is not needed. Also, Linux servers are widely used by many server hosts. This makes Linux servers highly available to the public. Because of the availability, server side security must be the main focus. Security maintenance will be factored in as a cost to the company.
3. **Storage Management**: For the client side, a mac server can be used in the local network to store backup data from the working mac computers. Mac servers work well with Mac computers, since Apple likes to develop their hardware to work with each other flawlessly. Also, mac servers are very secure. For the server side, a database will be used by the Linux server to save the data collected from the application. A database is useful to quickly query data. LDAP will be used to access information about a user quickly in a secure way. When querying a user account, LDAP will open a session for the user, and it will look up the information for the user.
4. **Memory Management**: For the client side in developing the application, MacOs has a basic memory management technique. The system finds free space or creates free space in memory. Then it sends a reference to that space in memory. For the server side, a database uses tables, records, values and keys. The table will hold records of specific data that is used for the application, i.e. Users. For example, the User table will hold the values name, username and password. Passwords are usually encrypted in the database by using a hash function.
5. **Distributed Systems and Networks**: An interface would be created to communicate between the server and the client. This can be achieved by a GET, POST, PUT or DELETE HTTP request. The server will authorize the request if accepted, and then it would return a JSON syntax so that any client, regardless of operating system, can read it. However, sometimes the server will not respond because it is updating or down. A timer in the client-side can be used to determine if the server is down or not responding.
6. **Security**: For the client side, MacOS is very secure. I also recommended saving data to Mac server. This will further the security of the data when the application is being developed. For the server side, I recommend Linux. Since Linux is open source, there are many security protocols that are developed that can be used to protect the server. This will help protect the client’s data from any breeches. Also for security, I recommend an authentication API to authorize a client session request from the server. This will prevent any unwanted session requests from the server. I also recommend encrypting passwords and other sensitive data in the database. If an attacker were to gain hold of the database, they still won’t be able to read the sensitive data.