

Draw It or Lose It

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design** 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 | 05/20/23 | Jonathan Warner | Initial Version |
| 2.0 | 06/03/23 | Jonathan Warner | Added Evaluation |
| 3.0 | 06/18/23 | Jonathan Warner | Added Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based, multiplatform game based on their current Android game, Draw It or Lose It. The goal of this project is to create a multiplayer game where teams compete to guess what is being drawn, utilizing a large library of stock drawings as clues. A game consists of four-minute-long rounds during which drawings are rendered at a steady rate and are complete at the 30-second mark. The team guessing has until time expires to make their guess, after which the remaining teams will each get one guess with a 15-second time limit. In an effort to efficiently manage memory and prevent duplicate instances of entities, singleton identifiers will be associated with each entity so as to distinguish between multiple teams and players and to be able to ensure only one instance of a game is running at any given time.

## Requirements

* Multiplatform support: The application should be developed to serve a multitude of devices and operating systems.
* Multiple teams and players: The application should support the ability to create and play with one or more teams, with multiple players assigned to each team.
* Unique game and team names: The application should ensure that the names of created games and teams are unique and cannot be duplicated.
* Single game instance: Only one instance of a game may exist in memory at any given time.

## [Design Constraints](#_2et92p0)

1. **Network Connectivity:** Applications developed in a web-based distributed environment rely on network connectivity. The constraint resulting from this is that the application’s responsiveness and functionality may be impacted by the quality and stability of the user’s network connection.

**Implications:** The application should be developed to handle connection disruptions gracefully and with little hinderance to the user.

1. **Security:** Applications developed in a web-based distributed environment need to be concerned with security, especially if any user data is collected in the future. Being connected to a network opens up avenues for attack if not properly guarded against.

**Implications:** The application should have safeguards to properly protect any user data that may be collected presently or in the future.

1. **Compatibility:** The Gaming Room’s desire for multiplatform support requires that multiple devices, web browsers, and operating systems are considered during development.

**Implications:** The development of this application should take into account which languages and practices provide the most compatibility with the desired platforms. Picking a language with proper multiplatform support could save the team time that would be spent coding this themselves.

## [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 following UML diagram demonstrates the structure our application will utilize. ProgramDriver is the class that will drive the program and thus contains main(). The SingletonTester class is simply a class that the driver will utilize to test if singleton variables are properly functioning. The GameService class serves as the framework that handles the current instance of the currently running game. Four classes are utilized to create games, teams, and players. The base class Entity will serve as the parent class for every created game, team, and player. Each game, team, and player will inherit basic attributes from the Entity class and then implement ones unique to themselves. The GameService class is associated with the Game class and can add games. The Game class is then associated with the Team class and can add teams. Finally, the Team class is associated with the Player class and can add players.

**"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)

| **Dev Reqs.** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac OS X Server is an Apple-made server software designed to be compatible with Mac OS. The software utilizes the popular web server, Apache. Macs are generally top-of-the-line in terms of hardware and have relative safety from malicious activities thanks to the safety the Mac OS’ Unix-based architecture provides. A drawback associated with Macs is that they tend to be expensive, which would make scaling up for larger-scale web-based applications cost-prohibitive. | Linux is an open-source, Unix-like operating system based on the Linux kernel. This operating system is cost-effective for web hosting when compared to both Windows and Mac. Linux is known to be flexible, which, when paired with its low cost, allows for increased scalability. Despite this, the software can be more difficult to set up and configure compared to other operating systems. Compatibility problems are also a common problem associated with Linux and need to be considered if third-party software is utilized on the server. | Windows, a popular operating system created by Microsoft, has a dedicated version for web hosting called Windows Server. The software is user-friendly and very familiar for individuals who mainly use Windows for their own systems. Windows Server was created with support in mind for enterprise-level management, data storage, applications, and communications. The costs of using Windows server-side would most likely be between Mac and Linux, which have the chance to impact scalability. Windows servers tend to also be less stable compared to other solutions, even if that has dramatically improved in recent years. The operating system is also more susceptible to external intrusions, such as viruses. | Mobile devices can be a convenient way to access web-based applications remotely and from anywhere, but they are poor choices for web hosting. While they have the ability to take advantage of cloud computing to help deal with resource costs, their limited processing power and storage capacity limit any possibility of hosting large, complex web-based applications. There is some niche software for web hosting, but generally, mobile devices should not handle the server-side and should mainly be considered from a client-side point of view. |
| **Client Side** | Developing software for Mac could increase costs by requiring additional expertise or licensing with regards to Apple’s development tools. This would inevitably add time if extra training were required for programmers. If getting the application onto the Mac OS Apple Store is a goal, its guidelines would need to be adhered to, possibly increasing development time. | Developing software for Linux is both cost-effective and time-efficient thanks to the availability of open-source development tools and libraries. Depending on the programmer, Linux may be unfamiliar and could require extra training to work with. | Windows is the most popular operating system, and most programmers are familiar with the environment. This familiarity could reduce development times or the need for extra training. Its popularity makes Windows a safe bet for being the first OS to design the application for. | Mobile development may require extra training depending on the team’s familiarity with that type of work. Supporting the various mobile operating systems and devices could bloat development costs, depending on how many are focused on. This support would also introduce an increased need for testing in a multitude of these environments. Guidelines or requirements for the multitude of app stores across different mobile platforms could be a major consideration for mobile app development and greatly impact development time. |
| **Development Tools** | Development for Mac OS applications typically takes advantage of two programming languages. Objective-C is an older, legacy code base that was and is still used to develop apps for Mac OS. A benefit of this language is its compatibility with the newer language that Apple has developed, Swift. This language is phasing Objective-C out and is quickly becoming preferred thanks to its more modern syntax and range of frameworks and libraries made for Mac OS development. Apple’s official IDE is called Xcode. It boasts high integration with the Swift language by utilizing SwiftUI. The cross-platform IDE Visual Studio also has a Mac version that is commonly used. | Development for Linux often utilizes low-level programming languages, such as C and C++. Both of these languages are powerful, but have higher complexity than an alternative, such as Python. Python is another common language when developing for Linux thanks to its versatility and ease of understanding. When considering web-based application development for Linux, JavaScript is a popular choice. Linux can use a variety of IDEs, but a popular one is called GNU Compiler Collection, or GCC. This IDE is a collection of compilers and libraries with support for a multitude of different languages. An alternative to this is an IDE called Vim. Vim is actually a text editor often used by Linux developers. It provides a command-line interface and extensive customization through plugins. | Three relevant languages for the development of this application are C#, Python, and Java. C# can utilize the .NET framework for Windows applications and includes a large library to work with. Python has the benefit of being easily readable and more simplistic, but at the cost of code efficiency. This language is extremely popular and also has a large number of libraries and frameworks it can work with. Finally, Java is a widely used language when it comes to cross-platform development. This is thanks to its use of the Java Virtual Machine (JVM). An IDE like Visual Studio, Microsoft’s official IDE, is a solid choice to work with any of these languages. If Java is the language chosen, Eclipse is an alternate IDE that could be a better choice. This IDE is open-source and has support for many different languages, but Java development is what it was created for and written in. | Developing an application with support for mobile devices in mind requires a language with good cross-platform development support. Possible choices include Java, Swift, and Kotlin. Java is a strong option with the Java Virtual Machine. The JVM could make porting a mobile application to different mobile operating systems easier. Swift would be utilized if Apple devices were the primary mobile devices being developed for or ported to. This restriction makes it the worst choice of the three for this project. Kotlin is a somewhat new programming language that has become the official language of Android applications. Developed by Google, Kotlin is preferred when creating applications for Android but is also effective at creating cross-platform apps with its built-in multiplatform support. Microsoft’s IDE, Visual Studio, can be utilized when working with either Java or Kotlin but does not have native support for Swift. Apple’s Xcode can be used for Swift development. Google also has an official IDE for Android app development called Android Studio. This IDE is intended to be used with Kotlin but can also work with languages such as Java and C++. |

## 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**: Based on the project’s requirements and our evaluation, Linux is our recommended choice for Draw It or Lose It’s server-side operating platform. This recommendation stems from the fact that Linux is extremely flexible, cost-effective, and highly scalable. It also has many options to pick from in terms of specific Linux-based operating systems that are focused on web hosting. One popular choice that stands out for us is Ubuntu Server. This software provides a secure and reliable foundation for hosting any web-based application and boasts the best value scale-out performance available.
2. **Operating Systems Architectures**: Our recommendation of Linux, or more specifically, Ubuntu Server, would follow a client-server model. The server side would be developed to handle Draw It or Lose It’s logic, communications, and data. Access to the server side would then be provided to the client side through a web browser. This connection is made possible by allowing the browser to interface with the game’s exposed REST API. Developing the game to be accessed through a web browser also ensures compatibility with different platforms.
3. **Storage Management**: For Ubuntu Server, our recommendation for storage management would be the relational database management system (RDBMS) MySQL. MySQL is an open-source software that allows for the creation and management of a database to store game-related data, such as usernames, drawing clues, or high scores. This software is easy to use and is meant to handle an immense amount of data, meaning it is also highly scalable. In addition to using MySQL for database management, a network filesystem could be utilized to manage unstructured data, such as the high-quality image files the game works with. GlusterFS, an open-source distributed file system, would be our recommendation here. The software allows for high-availability storage and redundancy by copying and replicating the data across more than one server.
4. **Memory Management**: Optimizations for memory management would happen largely in Draw It or Lose It’s coding, but Ubuntu Server has multiple features that help with efficiency. For example, since Ubuntu Server is a Linux-based operating system, we can utilize Linux’s file system caching to store frequently requested data in memory for faster access. The operating platform also has access to Linux’s virtual memory, which uses a disk as an extension of memory, increasing the effective size of usable memory. The kernel checks for unused blocks of data in memory and writes them to the hard disk, freeing up those resources and further improving performance.
5. **Distributed Systems and Networks**: In order to allow Draw It or Lose It to communicate between various platforms, we recommend the use of distributed software, such as GlusterFS, alongside the utilization of the REST API. As mentioned prior, GlusterFS is a distributed file system that can be utilized to spread data or databases associated with the game across multiple servers, which allows for improved efficiency and redundancy. The improved efficiency comes from being able to have different servers handle different aspects of the game. The redundancy comes from having data stored separately and in multiple places at once. If something fails on one server, it shouldn’t affect the others. Communication across these servers and with the clients’ web browsers should be done using a well-defined REST API. By utilizing a REST API and developing the game to be played in a web browser on the client-side, we can easily allow different platforms and devices to interact with the game servers.
6. **Security**: Security is an important consideration for a web-based game, especially with regard to user data. Ubuntu Server is secure right from installation and boasts a multitude of security features. These include password hashing, file encryption, and many more. A form of role-based access control (RBAC) should also be implemented. This will enable us to designate roles with different levels of access using the "least privilege principle". This principle states that each role should be provided with the least possible number of privileges while still allowing them to successfully perform their function. Utilizing Ubuntu Server’s file encryption in tandem with the implementation of RBAC should ensure user data remains secure. In terms of what the client will access on their end, a secure protocol should be used for communication with the servers. HTTPS, or Hypertext Transfer Protocol Secure, is our recommendation for this, as it sufficiently encrypts the data flow between server and client.