

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.3

## 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 |
| --- | --- | --- | --- |
| 1.3 | 10/17/21 | Shaun Ryan | Updated Evaluation and Recommendation sections. |

## [Executive Summary](#_sbfa50wo7nsh)

The social media company Chat Away operates a successful browser-based chat app. In an effort to expand their customer base and product immersion, the client wants to develop mobile apps to interface with their existing webpage. Their end goal is to increase not only the number of users, but also revenue.

## [Design Constraints](#_2et92p0)

Platform Constraints:

The mobile applications will have to be coded specifically for Android and iOS. This will require development teams knowledgeable of each platform. Despite the two apps being developed independently, they must have the same functionality, design language and branding.

Interface Constraints:

The mobile applications will have to interface with the existing Chat Away website. An API will need to be developed to allow both the iOS and Android apps to interface with the Chat Away database is a secure and timely manner. This will require technical knowledge of the existing Chat Away infrastructure.

Budgetary Constraints:

Chat Away is outsourcing the project with a goal of staying within a set budget. This will limit scope creep and time available for development and testing.

## [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)

Entity is created as a base class with Game, Team, and Player set up as child classes. From Entity they inherit attributes such as id and name, as well as behaviors like getId() and getName(). GameService is a class set up as a singleton so only one instance can exist at a time. The singleton GameService can be queried for a reference to its instance, which will be used by all players. GameService has a “zero-to-many” association with the Game class. It can instantiate any number of games and uses an iterator pattern to ensure that they are named uniquely.

The Game class has a “zero-to-many” association with the Team class which has a “zero-to-many” association with the Player class. The Game class can instantiate any number teams and the Team class can instantiate any number of players. Iterator patterns are use for each of these to check for and ensure unique names.

ProgramDriver is used as a testing class which retrieves the instance of GameService and initializes multiple Game, Team, and Player objects. It then uses SingletonTester, which also retrieves the instance of GameService and prints the list of game objects. This verifies that the instance of GameService used by ProgramDriver is the same as the one used by SingletonTester.

**"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** | Due to their higher cost, restricted hardware, and low market share, Mac/macOS would be a poor choice as a central server for any application that doesn’t require staying within the Mac ecosystem. | Having the largest market share for cloud servers, including the majority of AWS, Linux would be a sound choice to host the server for Draw It or Lose It. A Linux server would be able to host a Java application easily and reliably. | Although their market share is less than Linux in the online server space, a Windows server would be another good option for hosting the application server. It would provide a robust platform with native Java support. | While mobile devices will likely be the largest percentage of client applications, they should not be a consideration for the server-side hosting. |
| **Client Side** | macOS applications can be distributed directly from our client’s website or they can be added to the App Store. While the latter would take more time and has associated costs, it would provide for the largest visibility in the market. | Distributing an application for Linux requires some nuanced consideration. Without licensing the software as open source, we are unlikely to have our client added to official repositories. There is also the issue of which distributions are targeted. The most pragmatic option would be to offer a .deb and .rpm build, which covers Debian and Red Hat derived distros, covering a majority of the platform. | Creating and distributing a Windows client is much more straightforward by comparison. While we could go through the process of having it added to the Microsoft Store, it would have a negligible effect on the actual development from a cost, time or knowledge perspective. | Programming the client app for mobile platforms will be much more locked down from a development and distribution standpoint. Both iOS and Android apps would have to go through the process of being added to the App Store and Google Play Store respectively. |
| **Development Tools** | While macOS can run Java applications, and very little would need to be changed from the code used on other platforms, it may not be the optimal route. Most modern macOS applications are developed in the Swift programming language through Apple’s Xcode. This will provide the highest level of compatibility, performance, and security. However, it also requires familiarity with the language. | All Linux distributions have the ability to install Java and run Java applications. The code used for this application would not differ greatly from the code used for the Windows client or Mac client (if Java was chosen as a language). Creating precompiled binaries for multiple distributions would increase time and cost but would also increase the number of possible users. | Having the largest market share by a large margin in the desktop client space, Windows would be the main target for users. Java can be developed and run seamlessly on Windows. Backward compatibility won’t be a consideration, as the vast majority of users are on compatible versions of Windows (7 through 11). Eclipse or any other popular IDE would be a good choice for development. | iOS apps will need to be developed in Swift, which will require additional training and testing. If a macOS desktop client is already developed, some code may be able to be reused. For Android, the app will most likely be developed in Kotlin. As it is based on Java, the increase in training, cost and time should be minimal. |

## 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**: For the continued development of Draw It or Lose It, I would recommend that The Gaming Room starts with Linux devices. While this is a significantly smaller percent of the market than Windows or even Mac, there are advantages to targeting Linux for the initial development before branching out to other platforms.
2. **Operating Systems Architectures**: The various distributions of Linux offer modern features, on par with competing platforms. They have graphical user interfaces (GUIs) with robust features and ample documentation. They also offer access to command line terminals where administrators have unparalleled abilities to configure the system to their needs. Development can be facilitated in a variety of IDEs, such as Eclipse, VS Code, Netbeans, and multiple JetBrains products.
3. **Storage Management**: Storage through Linux works similarly to other operating systems. The application will be able to store files where the user of the application has write access. This could either be on a local drive or to a cloud-based system. Outside of locations for storage, there are little other limitations on how the data can be written and read. All of the major desktop environments (e.g. KDE, Gnome, XFCE, etc.) have APIs that allow the application to use their file management applications, offloading the burden from The Game Room developers.
4. **Memory Management**: Linux offers multiple methods of creating and distributing packages. In recent years, “container” style packages have grown rapidly in popularity. The Flatpak packaging format, for example, contains the precompiled binaries and all necessary dependencies included. The biggest advantage of these formats is that they are completely sandboxed. The memory used is protected from other applications running on the system. This will increase security and reliability.
5. **Distributed Systems and Networks**: Network access is another area where a Linux based client would be just as capable as any other platform. HTTP and HTTPS protocols are supported out of the box, as well as other common network protocols. To maximize uptime of the server, a hosted service such as Amazon’s AWS or similar services from Google, IBM or Oracle would be advantageous over hosting the server “in house”. On these services setting up a LAMP (Linux, Apache, MySql, PHP) stack is trivial and often preconfigured by default. This will allow The Game Room to migrate their Draw It or Lose It server easily and maintain their current Jersey/HTTP implementation.
6. **Security**: Security is one area where Linux is a class leader among potential client platforms. User accounts have strict regulations on what users can access and change. Keeping users and applications at the lowest needed permissions is a guiding principle of the system. Elevating privileges is a much more deliberate act than on platforms such as Windows (which the user can do without realizing the implications). Sandboxing applications, as described earlier, is another method for increasing security from other applications which could potentially be malware or spyware. Built in support for virtual private networks (VPNs) is one more technique for increasing security of files shared over a network connection.