

<Draw It or Lose It>

# **Web Game Application Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_gjdgxs)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_3znysh7)2

[**Executive Summary**](#_2et92p0)3

[**Design Constraints**](#_tyjcwt)3

[**System Architecture View**](#_3dy6vkm)3

[**Domain Model**](#_1t3h5sf)3

[**Evaluation**](#_4d34og8)3

[**Recommendations**](#_17dp8vu)5

## [Document Revision History](#_3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.2 | 12/10/2022 | Brent Artuch | Recommendations Update |

**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](#_2et92p0)

The Gaming Room desires to create a multi- platform version of their android game “Draw It or Lose It” which is currently only available as an Android application. The game will consist of teams of playes who need to guess a rendered drawing until it is completed after thirty seconds. The game will consist of four, one-minute, rounds, and if the team does not guess correctly before time expires, than the other teams each have fifteen seconds to make a guess. To begin, we will need to utilize the Java programming language, because it is highly portable across most platforms. Also, because the game has been made for Android, the source code should be easily transferred over to a web application. This is due to Android apps being mostly written in either Java or Kotlin combined with XML for the GUI. We can then use tools such as JavaFX and JPanel to create the GUI for the web-based version of the application. We will also need to implement a few different design patterns along with Object Oriented Principles to meet the requirements outlined in the next section.

## [Design Constraints](#_tyjcwt)

1. The game must be available across multiple platforms. Java will be the most efficient choice of programming language because of its portability and close relationship with the Android development of the original version of the game. To make the game available for iOS, we will need to translate the program to the Swift or Objective C languages for compatibility purposes.
2. The game will require the ability to have one or more teams involved. To accomplish this, we will have to use Object Oriented Programming to create multiple team objects that will be added to the list within the game object after they’ve been constructed.
3. Each team will have multiple players assigned to it. Within the team object we will have a mutable list of players. There will need to be a player object with a constructor to create each unique player object.
4. Game and Team names must be unique, and the user should be allowed to check when a name is in use. We will utilize the iterator pattern to check these lists for an already existing name value.
5. Only one instance of the game can exist at any given time. The player, team, and game classes will all have unique identifiers for their class objects, and we will use the singleton pattern to ensure that there is only one instance of each of these objects.

## [System Architecture View](#_3dy6vkm)

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](#_1t3h5sf)

In regard to the UML diagram, the ProgramDriver will contain our main function for the application and will call the SingletonTester testSingleton method to confirm that only one instance of the GameService class exists in memory at any given time. The main function will only contain objects and references to their related methods to increase readability. The Entity class will be a parent class with three corresponding sub-classes of which are Game, Team, and Player. The sub-classes will inherit attributes from the parent class through overridden constructors giving the derived objects of each class unique name and id identifiers, and in doing so, will reduce the amount of code needed to complete each class’s functionality. The constructor for GameService is a private method to prevent any additional instances outside of its own class. The three subclasses as well as the GameService class have aggregation relationships, because GameService will have a list of game objects (0 to many), Game will have a list of team objects (0 to many), and Team will have a list of player objects (0 to many). Because of this setup, teams for the games will have players, the game will have teams, and the GameService class will be able to get and validate the information from the other classes.

**"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](#_4d34og8)

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** | Advantages:  Utilizes UNIX which is great for web development because most web apps are produced using UNIX/Linux.  Has access to a variety of open-source software at terminal level.  Wide array of development tools for creating and maintaining server software.  Server deployment is integrated into macOS with documentation available. Cost is around $20 for usage  Disadvantages:  The hardware is very expensive.  Lack of support for the MacOS. Many applications aren’t portable to the OS.  Discontinued server in app store. | Advantages:  Open-source with a dedicated community of developers.  Provides encryption and has strong network security.  Source codes and scripts run fast and efficient compared to other OS.  Tivoli server available through IBM with documentation. Cost is around $400 for usage  Disadvantages:  More prone to system crashes because of the open-source format.  Limited tech support from developers because it is distributed for free. | Advantages:  Wide range of software and support available.  Most widely used OS.  Moderate affordability falling in between the other two desktop OS choices.  Utilizes WSD server deployment with a 2019 version available and free online training. Pricing starts at around $500.  Disadvantages:  Over customization  can lead to crashes.  Majority of Windows software costs money and not a whole lot is included. | Advantages:  High levels of availability with use of cellular networks to reach users.  Use of cloud storage to access servers across multiple devices.  Disadvantages:  Need to utilize one of the three desktop operating systems to develop the databases.  Semi-strict guidelines for web apps on Google Play Store and more so on the iOS App Store.  Required to utilize two different mobile development IDEs for the development process. |
| **Client Side** | Cost: Will have a moderate ranged cost to the client because many applications are included in the app store for Desktop devices.  Time: Will have a low amount of time required because of the high-quality responsive hardware in Mac devices and simplified/fluid UI.  Expertise: Low level of client expertise required to navigate operating system because of simplified interface and multitude of learning tools provided by apple integrated into system. | Cost: Linux comes with a lower cost to the client because of the open-source applications available.  Time:  Development will take considerably more time because of the depth of technical knowledge required to navigate the Linux OS and Applications.  Expertise: not beginner friendly and will require a knowledgeable client to navigate the OS effectively. | Cost: Windows will have the highest cost of the desktop OS to the user due to most applications not being included.  Time: Moderate level of time required to navigate OS and research for information on support channels for OS related errors.  Expertise: Moderate level of expertise required for usage due to extensive UI options/customization and research required to troubleshoot OS related bugs/errors. | Cost: Mobile devices will require Mac hardware to support iOS development on top of Android development, so this corelates to a higher expense.  Time: Will take the highest amount of time because of the need to write the apps in both Kotlin and swift, in addition to the UI languages associated with them.  Expertise: Will acquire the highest level of developer expertise because of the need to develop the web app for both iOS and Android platforms.  Will give the users the highest amount of access because of the frequent use and mobility of Android and iOS devices. |
| **Development Tools** | Web Browser:  Safari  Primary IDE:  XCode  Cloud Service:  iCloud and XCode Cloud  Unique Languages:  Swift and AppleScript  API:  Web Extensions  Universal Desktop Backend Languages: Java, C++, Python, etc.  Universal Desktop Markup Languages:  HTML, CSS, and JavaScript  One development team required to handle that specializes in the use of swift and swift UI | Web Browser: Chromium and Firefox  Primary IDE:  Vim  Cloud Service:  pCloud, Ubuntu, and Mega  API:  Many options because of open-source formatting. Postman, RESTAssured, etc.    Primary Language:  C programming  Universal Desktop Languages: Java, C++, Python, etc.  Universal Desktop Markup Languages:  HTML, CSS, and JavaScript  One to two development teams required to handle the added complexity of the Linux OS. | Web Browser:  Microsoft Edge  Primary IDEs:  Visual Studio and Visual Studio Code  Cloud Service:  Azure  Unique Languages:  Typescript and Xamarin  API:  ASP.NET Web APIs and Microsoft Graft API  Universal Desktop Languages: Java, C++, Python, etc.  Universal Desktop Markup Languages:  HTML, CSS, and JavaScript  One development team needed specializing in the use of visual studio IDEs and Microsoft frameworks. | Android:  Web Browser:  Google Chrome  IDE:  Android Studio  Language:  Java, Kotlin, and XML  API:  Firebase  Cloud Service:  Google Drive  iOS:  Web Browser:  Safari  IDE:  XCode  Language:  Swift and SwiftUI  API:  Web Extensions  Cloud Service:  iCloud and XCode Cloud  Two to three development teams required to develop Android version, iOS version, and to ensure that both versions provide a uniform user experience. |

## 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**: mac OS Ventura 13.0.1 Apple Silicon
2. **Operating Systems Architectures**:

* Unified memory architecture: CPU and GPU run on unified memory resulting in shorter distances for data to travel which leads to faster memory for both processing units. This results in no overhead because it becomes unnecessary to copy data across a PCIe bus.
* Asymmetric cores: Otherwise known as AMP systems, utilizes performance and efficiency cores to perform tasks that are best suited to each (i.e., background apps would run on E cores and a foreground game app would run on P cores).
* Grand Central Dispatch: Used to accommodate the needs of applications running on the system to appropriate hardware resources. Include APIs within GCD such as concurrentPeform can help with distributing work across the CPU cores with proper load balance over the proper number of iterations.
* Rosetta 2: translation process for Intel-based x86\_64 apps to be executed on ARM-based Apple silicon (including JIT compilers).
* Native Applications: applications can be ported between Intel-based architectures and Apple Silicon.

1. **Storage Management**:

* Google Cloud Service: Allows for storage regardless of geographical location or platform and will be shared across all devices associated with logged in accounts. Storage amounts are nearly limitless but require payment for expansion.
* Hard backup: Keep SSD storage backup of application versions incase of any corruption of Cloud Files or loss of source data due to human error by administrators of the game application.
* Finder - Mac file management system that will organize the game application files for the hard back up in addition to the Google Cloud file system linked to the administrators account.
* Firebase SDK (Mobile): A powerful, simple and cost effective storage service that works with Google Cloud to store user data for mobile and web based applications regardless of network quality and with high levels of scalability. It is compatible with both Android and iOS.
* JSON API (Desktop): Type of RESTful API that is backed by JSON for web development with Google Cloud solution, and is fully compatible with Cloud Storage Client Libraries. Utilize JSON.simple library for preparing JSON data in Java.

1. **Memory Management**:

* Allocation: Determine how much memory will be required for the Draw It or Lose It application to temporarily store data.
* First fit algorithm: Will quickly assign data to the first available block of memory that is sufficient size for the application to temporarily store cache data items. If a large portion of the block is unutilized, it will split to free up memory to be used elsewhere.
* Java garbage collection: Allows for high levels of memory efficiency because it will automatically remove unreferenced objects from the heap to create space for active objects to use.
* Java Portability: The java application will be portable across all of the platforms that the client desires to utilize for the app with the exception of partially of Android and fully of iOS. The source code will need to be adjusted to work with XML in Android Studio and rewritten completely in Xcode/Swift for iOS.
* Activity Monitor: Mac application that provides graphical feedback of memory usage by the system and its applications including memory efficiency, cached files, and swap usage.

1. **Distributed Systems and Networks**:

* Google Cloud Service: Web application will be linked with Google Cloud for storage of game data. This is the most ideal solution because the likelihood of connectivity issues or outages is low compared to a physical server because servers can crash or lose connection much easier than a cloud service.
* Java and Maven: We can utilize a Maven project in Java to create a web version of the application that will be portable across all desktop (or Mobile platforms if the users opts to use the browser version) platforms via HTTP requests.
* RESTful API: JSON API will link the Draw It or Lose It application to the cloud libraries where the saved game data can be stored and accessed for future use.
* Client Side UI: User Java GUI will be created utilizing Swing or JavaFX to create a visually appealing game environment for the user to interact with at login and throughout execution of the game application.
* Swift and Kotlin (Mobile): Mobile devices will be written in their respective languages but will be able to share the Firebase SDK (part of the Google Cloud service) as a data server for authentications, login credentials, and user saved data.

1. **Security**:

* Write XOR execute: Security feature for the server side of the application to help prevent disruptive changes from being made to the source code as well as other elements by only allowing memory to be writable or executable but not both. Apple silicon excels in this category because of the quick toggle features in the OS that allow for rapid changes between the two memory states.
* Kernel integrity protection: System Integrity Protection (SIP) allows macOS to prevent running of unwelcome applications or code within the system. The system will only allow code to run if you explicitly give it permission to do so further protecting the server side of the application from possible intrusion.
* Authorizations: Each user will have to go through an authentication process to access their account by using proper credentials and two step verification. The user will be prompted by the login interface to update their password on a periodic basis as to prevent intruders from guessing their passwords.
* Defense in Depth: Analyze the application and server connections and activities regularly for any abnormalities that may be present and implement alert systems that can flag possible intrusions automatically. Conduct regular penetration tests on the application and server connections to see first hand if there might be any holes in the server’s security.
* Client Server Architectural Pattern: With this architectural pattern, the user will create a login credential profile when first playing the game to be stored in the cloud service along with any save data for the game. The profile will then set permissions for the user’s account, allowing them access only to portions of the program that they have proper credentials to interact with.

Recommendations References:

1. cloud.google. (2022, December 5). *Cloud Storage*. https://cloud.google.com/storage/docs/
2. developer.apple. (2022). *Documentation.* <https://developer.apple.com/documentation/apple-silicon/>
3. firebase.google. (2022, December 5). *Cloud Storage for Firebase.* <https://firebase.google.com/docs/>storage/
4. Gagne, G., Galvin, P. B., and Silberschatz A. (2022). Operating Systems Concepts, 8th Edition. Wiley. O’Reilly Media, Inc.
5. Ravenbrook Limited (2020). *Memory Management Reference*. <https://www.memorymanagement.org/>mmref/begin.html#:[~:text=1.3.-,Application%20memor](https://www.memorymanagement.org/mmref/begin.html#:~:text=1.3.-,Application%20memory%20management,it%20is%20no%20longer%20required)y%20management,it%20is%20no%20longer%20required.
6. support.apple. (2022). *View memory usage in Activity Monitor on Mac*. <https://support.apple.com/>

en-am/guide/activity-monitor/actmntr1004/mac