

Draw It or Lose It

# **CS 230 Project Software Design Template**

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 | 09/17/2022 | Morgan Fox | Executive summary, design constraints, domain model |
| 2.0 | 09/30/2022 | Morgan Fox | Evaluation |
| 3.0 | 10/14/2022 | Morgan Fox | Recommendation |

**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 is an app developer responsible for creating the Android app Draw it or Lose It, based on the ‘80s game show Win, Lose or Draw. The client has approached Creative Technology Solutions to help them in porting their mobile app to a web-based application capable of running across devices and operating platforms. The game involves one or more uniquely named teams, each with multiple players. Game names must be unique and only one instance can run at the same time. Unique objects representing the game, team and players will be used to implement this requirement.

## [Design Constraints](#_2et92p0)

* The web app must be able to pull from the same stock photo repository as the mobile app
* The web app should run well on a variety of platforms, including iOS web browsers
* The web app should use as much cross-platform code as possible to improve maintainability
* The app should utilize modern web standards to future-proof it against deprecated features
* The app should practice contemporary cybersecurity procedures for similar apps
* The app must contain multiplayer functionality
* There should only be one instance of the game running on a device at a given time
* The program needs to be built in Java

## [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 Entity class is inherited by the Game, Team and Player classes to encapsulate metadata (such as ID and name) about the object that is common across all three structures. The Team class contains references to the Player class, and in term the Game class references the Team class and the GameService class references the Game class.

This form of encapsulation modularizes the code by containing attributes and methods responsible for governing specific application functions within the same discrete classes and objects. This delegation also allows for abstraction, with the methods we design serving as interfaces for internal object data, thus making the code easier to update in as few places as possible.

**"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** | Apple has discontinued its MacOS Server product as of this year, thus the lack of long-term support makes it less than ideal for this project and it will not be considered. | Linux has the popular advantage of being free and open source (FOSS) software, meaning that it can be implemented at a lower cost.  Linux servers are also more stable and less prone to virus infections due to its decreased prevalence as a consumer and office system and a strong open source community. It’s also fairly easy to port applications to Linux even if you develop on other platforms.  On the other hand, Linux is less user-friendly and depending on the distribution may not have the same dedicated support that Microsoft can provide. Thus, it’s not as easy to setup, though its overall stability does mean it requires less maintenance.  Cost: $0+ | Windows servers have the benefit of dedicated tech support from Microsoft and a near guarantee of Microsoft constantly developing new, improved server OSes.  However, Windows is not free. Its proprietary nature also can make software compatibility worse in some ways, as fixes for issues can’t always be implemented in a timely fashion.  Windows tends to be easier to setup than Linux, but generally speaking it has worse performance. As hard as Microsoft tries to improve it (which is little, because it has a financial incentive to sell future versions), it’s still got lots of bloat. It’s also an easier target for bots, hackers and other bad acts.  Cost: $501-6,155 | No. Just no. A mobile OS is optimized for consumer usage. The devices they run on have worse performance than equivalently-priced desktop servers.  And they will have poor software compatibility. Unless jailbroken, they have limited access to manipulate like a real server OS.  And if we’re running it on an actual mobile device, the network will be insufficient.  To reiterate: this is a bad idea and will not be further investigated. |
| **Client Side** | Java is available on MacOS, and works with major browsers available on it (including Safari, Chrome and Firefox).  There are no special tools needed for developing Java applications that run in a browser on Mac, however, steps could be taken so that a Java app looks more like a native app on Mac, which would benefit usability for Mac users. That is, of course, assuming that the smaller market share of Macs is worth investing in. | Java is available for Linux and the browsers (like Firefox and Chrome) available on the platform. Linux has no issues connecting to the web.  However, the market share for Linux-exclusive consumers is pretty low, thus calling into question whether it's worth specifically supporting it. | Like Linux, Windows software can be written in everything from C++ and C# to Python and Java. It's the most-used PC operating system in the world and thus has a very attractive market share.  It’s also easy to develop for and wildly popular. | Java is natively supported on Android, which the app is already available on. To run on iOS, however, it’d need to be ported in a different programming language like Swift, which is Apple’s new coding language developed for Mac and iOS.  Developing for Android requires only an Android phone for testing. Developing for iOS, however, requires procuring Macs, an iPhone and a programmer who can write in Swift. All for a platform that has a minority market share but a majority app income share.  Apple also has more stringent app guidelines for App Store than are on Android app stores. Thus, special care and attention must be paid to the iOS app guidelines to ensure compliance, else the app risks being rejected. This also means there’s less room to style it however we want, as iPhone users expect their apps to have a similar look and feel. |
| **Development Tools** | We don’t believe that any special development tools are required to develop a browser-based Java app that runs on MacOS. We can use the same tools we do for Linux and Windows.  If we do develop on MacOS, Xcode is a popular IDE for the platform that supports Java, Objective C, Swift and other languages. Xcode is free.  Development requires installation of the Java Development Kit. | Again, as we’re developing a browser Java application, we don’t need to make special considerations for developing for Linux.  Linux has a large programmer community and many Java developers favor Eclipse, IntelliJ or Android Studio. Some IDEs charge various fees or subscriptions for professional use.  Development requires installation of the Java Development Kit. | Windows is the most popular operating system and can run a variety of IDEs that support Java, including Eclipse, IntelliJ, Android Studio, Visual Studio and VS Code.  There is no fee to develop a browser-based Java application for Windows. However, many commercial IDEs charge a fee for professional use.  Development requires installation of the Java Development Kit. | iOS requires Xcode, Swift and a $99/year license to publish on the App Store. It also requires the use of a Mac PC to develop on.  Developing for iOS will require upfront costs above those of other platforms, such as hardware, software and Swift coder(s).  Android uses the Android SDK which is Kotlin-based, though it also supports Java and C++.  Google also offers the Android Studio IDE which is based on IntelliJ, though it is not strictly required.  Publishing on the Google Play Store costs $25/app. |

## 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**: Draw It or Lose It is already available on the Android operating system. Porting it to both iOS and the browsers will allow the game to expand to iPhones, iPads, Windows, Linux and MacOS computers. Building the game in Java and, for iOS, Swift, makes for an ideal move as the former especially can reuse code across operating systems that will run natively in web browsers.  
     
   On the server side, I recommend a LAMP (Linux, Apache, MySQL, PHP) stack. Specifically, I recommend the Ubuntu Server operating platform. Linux is good for interoperability because it’s the industry-standard, providing developers with time-tested, open-source tools for optimal server management. It also has lower licensing costs compared to a Windows server as well as improved security and reliability. The Ubuntu distribution offers paid support, long-term updates and other benefits compared to some other Linux distributions.
2. **Operating Systems Architectures**: Using a Ubuntu 64-bit server reduces the chances of performance or memory issues due to the larger memory access provided, as a 32-bit system is limited to 4 GB RAM and a 64-bit system is not. The client-server dichotomy will be implemented with the server handling game state and user data management, serving the appropriate text and images to the client, which may be implemented as a Java app on Android, a Java app in the browser or a Swift app on iOS.  
     
   This setup allows for the faster server to handle the more complex operations and prevent cheating by enforcing consistent game state between connected clients, while the potentially slower devices simply send input through the application interface and update based on callbacks received from the server. This frees up the client to run smoothly and the server to handle the large amounts of data processing required for multiple game instances running. Since the game is asynchronous, it is not necessary for the server to have top-tier speeds and latency, but they must be good enough to not delay the gameplay or create “rubberbanding” type issues.
3. **Storage Management**: Traditional hard drive discs will provide cheaper storage over solid state drives. While HDDs are theoretically slower than SSDs, The Gaming Room’s app is dealing with a minimal amount of data, namely 8 MB images and user records. User records can be stored in a MySQL database interfaced with Java. Images can be sent raw or compressed, though the latter does require some more CPU usage, which would also be competing with game state management, networking, and so on.  
     
   Since user data is relatively small and the images aren’t particularly precious, I recommend using multiple drives in a RAID 1 configuration for fault tolerance. 2 or 3 drives should be sufficient. Data should be organized into two folders: one for images, and one containing the database of user records. If user records contain sensitive information, the database should be encrypted.

1. **Memory Management**: The Ubuntu kernel was developed for use on multiple hardware architectures and as such memory management is split into three parts: zone\_DMA contains the first 16 MB of direct access memory for use by certain hardware devices, zone\_NORMAL contains the next 880 MB of memory, and zone\_HIGHMEM contains the rest. Each memory zone in the Linux kernel is managed by its own page allocator.  
     
   The modern Android operating platform uses the Android Runtime virtual machine to manage memory usage and contains automatic garbage management. It uses shared paging to reduce RAM overhead requirements by distributing resources across multiple applications. However, the garbage collector will not page out any resources unless the object reference is released, after which the ART will return it to the heap.  
     
   iOS also features automatic garbage collection at compilation in Xcode. However, it’s less lenient than the Android Runtime, so care should be taken by the Swift developers to reduce memory usage and be sure to manually release resources that are no longer in use to prevent memory leaks and program crashes.
2. **Distributed Systems and Networks**: All clients should connect to the server, which can create game instances shared by players. The server needs to be able to create and store references to individual players and ensure that they are assigned to the correct games. Since this is an online game, its servers should be hosted by a reliable hosting company and the clients should be able to account for and provide feedback on server outages or disconnections. In either case, the client should regularly attempt to ping the server and re-establish connection. Similarly, the server code should react when a player disconnects and attempt to re-establish connection before closing their instance.  
     
   The server has minimal bandwidth requirements, since it should only be called when players create or join a game, send their moves (which the server will send to other players) and end the game.
3. **Security**: Linux servers are more secure than Windows servers, but there are still necessary steps to take to keep them secure. First, configure a firewall to block traffic from unused ports and IP addresses from certain hostile states like North Korea, Iran and Russia. Second, half antivirus run periodically on the server to detect unwelcome visitors. Given that we have a pretty full understanding of what files should and shouldn’t be on the server, we should be able to easily detect files that shouldn’t be there.  
     
   The principle of least privilege will be implemented to prevent unauthorized entities, whether they be users or game objects, from accessing parts of the server they shouldn’t. The Java app should also be limited in terms of its read/write capabilities, as it shouldn’t need to access any kind of server configuration files. Thus, the Java application shouldn’t need ADMIN-level access. Also, user data should be sanitized to prevent SQL injections and other malicious third-party code exploits.  
     
   Traffic will be encrypted using SSL and TLS 1.2 to ward against man-in-the-middle attacks. Since the game is already a mobile app, and mobile users are likely to use some sort of public internet access point, this will help keep the client-server connection secure.