

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

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

**Document Revision History** 2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

**System Architecture View** 3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/13/21 | Benjamin Strauss | First edition |

**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)

Create a web-based game. This game should be functional on multiple platforms. The design should be based on the existing design. Currently Draw It or Lose It is only available as an Android App. Create a web-based version of this gaming app.

## [Design Constraints](#_2et92p0)

Create a large library of stock drawings (a phrase, title, or thing). Render these images over 30 seconds. If the guessing team correctly identifies the picture (by matching strings), that team receives a point. If that team cannot guess it after 30 seconds. The other teams have 15 seconds to steal by offering 1 guess each. After another 15 seconds, the next round begins. Games last 4 rounds. Games can have 1 team or many. Each team has multiple players. Game and team names must be unique. Only one instance of the game can exist in memory at a time.

## [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)

It is important to remember that this project requires the use of singleton. The program driver uses a singleton tester to make sure that only one game service exists in the memory of the user’s device. The game class, team class, and play class inherit from the entity class. The game class and the gameservice class are related; there can be no games in the gameservice or there can be many games in the gameservice as the user can open the site and not join a game or the user can play game after game.

The UML diagram uses encapsulation: public methods allow access to private data fields. This keeps the data secure. It also uses inheritance to follow DRY principals (Don’t Repeat Yourself). It also uses polymorphism in entity to overload the constructor.

**"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** | Macs are actually UNIX based OS and javascript is always up to date on Macs. A Node .js or an Apache web server can be run from a Mac. Although the OS of the server does not need to match the OS of the website Macs are the least common. | Linux is an excellent choice because it has simplified security features which are easy to use and doesn’t rely on a third party such as Microsoft or apple. The backend server for a distributed application should be run on a shared UNIX or VMS system. | A Windows desktop would be good option. A static public IP and a firewall would be needed. An option is to set a server up through Microsoft’s IIS (Internet Information Services). This can be used with simple HTML. | It is possible to host websites on iPhones. One option is to install iSH Linux Shell. This requires Testflight first. |
| **Client Side** | The software could be written as a Mac OS application. SwiftUI would be used in this case. Swift is the language that powers Mac OS and IOS apps. | The application could be adapted to Linux. However, not many users are on Linux. | It is important that the software is adaptable to windows | We recommend that the game adapted to mobile devices and distributed on the apple and android stores |
| **Development Tools** | Eclipse IDE. Java. Runnable jar files. Swift | Eclipse IDE. Java. Runnable jar files. Linux is relatively easy to write code for. C and GTK+ for GUI applications. | Eclipse IDE. Java. Runnable jar files. C++ | Eclipse IDE. Java. Runnable jar files. Swift. Js. Swift for iPhones. Android studio for Androids, it uses Java. |

## 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 largest leap that we can take is to create expand the app to Apple devices. A large part of the population uses iPhones. In addition, many of the customers with iPhones have expendable income. The challenges posed by coding for an Apple device are many but the payoff could be huge. In addition, once we code for iPhones IOS it is an easy step to go to Mac OS if desired.
2. **Operating Systems Architectures**: the most popular architecture for IOS is the model-view-controller pattern (or MVC). This pattern divides software constructs in terms of their responsibilities: data, tables, and inputs respectively. These three divisions will pass information to each other. OS architectures are blueprints for our developers to solve problems. The user interacts with the view, the passes it to the controller to deal with, if needed the controller will make changes to the model, which will change its data values, the controller then sees these changes and sets them for the view (brightdigit.com).
3. **Storage Management**: Storage management for IOS revolves around iCloud. iCloud backs up the app’s data daily. Only user-created documents should be stored in the folder named <Application\_Home>/Documents directory. This folder is backed up to iCloud daily. The size of the data stored here should be minimized so that the application does not take up an inordinate amount of space on the user’s icloud. Data that can be regenerated should be stored in <Application\_Home>/Library/Caches. This includes files that could be important if the user deletes and then redownloads the app. Cached data helps load the app each time it is opened and helps the app run because the cache’s files are easy to retrieve. Data that is stored temporarily, such as files regarding each game’s processes will be stored in <Application\_Home>/tmp. These files will be deleted after each game. Information such as game result and the people who played will be saved in a data field sent to icloud daily. Some data might be marked as ‘do not back up’ so that they are available offline. If we decide to use this feature we must also include a process for the user or the app to purge these files (developer.apple.com).
4. **Memory Management**: RAM is short term memory. The maximum RAM for an app on the Apple store is 5 GB. Apple now uses Automatic Reference Counting (ARC) so we don’t have to retain and release objects. ARC counts how many pointers, references, and handles are processed. We need to make sure that we don’t free or overwrite data that is still in use. This results in memory corruption and can cause the app to crash or corrupt user data. We also need to free data that is not in use so that memory leaks do not occur. Allocated memory must be freed if it is not going to be used again. Memory leaks can result in poor system performance or the app to terminate (shut down) (tutorialspoint.com).
5. **Distributed Systems and Networks**: Because the game will be run on Android and IOS devices it will be important that users on each are able to interact with each other. Since this game will be on multiple computers it will need to communicate via a network. This is a combination of the backend and the frontend. Only the front end is downloaded to the user’s device while the backend is the main processing capability. The front end handles user interface functions (kb.iu.edu).
6. **Security**: More than a billion devices use IOS apps so security is very important. IOS 9 forces apps to only connect to secure networks using HTTPS protocols and TLS1.2. The ATS (Apple Transport Security) does not use HTTP because it may not be secure. We also have to watch out for MITM (Man In The Middle) attacks. To do this we use SSL Cert pinning. SSL works on a chain of trust and creates an SSL certificate that is checked each time the user or app connects to the server. The certificate is trusted by the SSL Certificate Authority. It is also important to store info in KeyChain rather than NSUserDefaults. NSUserDefaults saves information between launches and is bad because it saves info in plain text in plist. KeyChain, an encrypted container, is better because it is encrypted. Xcode Config files should be used for any secret info instead of just leaving it in the codebase. One example of secret information is API keys. Tools such as Vault would also work to secure info. We also need to check for jailbroken devices before the app starts up each time. Also important is to make sure debugging information is hidden from users because it could reveal secret information.

We also need to be careful of corruption from third party apps or libraries, file data protection such as NSFileProtectionCompleteUnlessOpen, and protection against third party screen recordings or screenshots (medium.com).