

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 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 | 01/22/2023 | Matthew Taylor | Executive Summary, Design Constraints, and the Domain Model sections filled out. |
| 1.1 | 02-10-2024 | Matthew Taylor | Evaluation section filled out. |

**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 wants their Draw It or Lose It application that is already an Android app to be written so that it can be run as a web based application and be able to serve multiple platforms. The app needs to allow for multiple games and multiple teams that need to be unique. The app also needs to maintain a library of drawings for the teams to guess at what is being drawn. The Gaming Room needs to know how to set up the environment.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.
* The game needs to accommodate multiple platforms

## [Design Constraints](#_2et92p0)

The program will need to be developed separately for each desired platform. If the app should be web-based, on Android, and on iOS, all three will need to be developed using their respective development environments, or a tool such as React Native could be used for faster and cheaper development that encompasses all platforms.

The API should be developed in a way that all the different platforms that need to access it can do so.

Game and team names need to be unique.

Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [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)

There are pretty much all of the OOP principles at work here in this UML diagram.

Inheritance is shown by the Game, Team, and Player classes inheriting from the Entity Class. All of the different entities in the application need to have a unique identifier and name. These attributes can then be written as part of the Entity class rather than writing them three times, once in each of the other three classes.

Encapsulation is shown here as the UML diagram does a good job keeping the attributes and methods that don’t need to be accessed by other classes private. This way the data is better protected.

Polymorphism is shown to a degree by the default constructor for the Game, Team, and Player classes being overloaded ensuring that those objects can be created with their id and name properties already given values.

This setup is also portable. If another application or another part of this application needed to use any piece of this, it would be simple to grab the needed classes and reuse them.

**"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** | Mac offers Mac OS X Server as their web server solution. It is based on Apache which is a really popular web server. The server is very intuitive and easy to configure, but this option can be very expensive especially with the limited expensive hardware options available. | A linux server is very configurable and requires very few resources which makes it very cost effective. It is also compatible with a lot of different systems, and it is open source which means that it can be maintained and supported by the community. This open collaboration can lead to really great products that are also more secure in some ways. | Windows Server is known for its ease of use and for the security features which allows people that are less knowledgeable to use it. However, it is more expensive and less customizable than Linux. | You could code the backend on the mobile device and host it, but this is the least viable option. There are not nearly so many tools available to help, and a mobile device is not equipped to handle large amounts of traffic. |
| **Client Side** | The client side development for Mac is usually more expensive mostly for the costs associated with purchasing the devices. Someone with expertise in developing for Mac would be needed as well. Easy of use and development are both similar to windows, but it depends on what someone is used to. Users don’t need much time to adjust. | Linux is great for the development of the software as it is really cost effective, open source, and highly customizable with many different tools available for development. However, most every day users don’t use Linux and therefore would need encounter a learning curve when wanting to use the application discouraging them from doing so. | Windows has many tools that are unique, and can only be accessed on a windows machine (virtual or otherwise). Windows has tons of support for web-application development. | The convenience of having access to the application from anywhere would be a huge draw for many people. However the development is a bit more difficult in the need to account for dozens of different screen sizes and for the differences between Android and iOS. The two different operating systems can pose some additional challenges, but there are tools to help with that. |
| **Development Tools** | Visual Studio Code, Xcode, Homebrew, bash command line (or I like to use Zsh), iTerm2, Tower, Javascript, HTML, CSS, React, react-native, react-native-web, Python, Java, Ruby, Rails, Spring, Flask, eclipse, npm, yarn, MySQL, Chrome, git  Typically one team is required for development for Mac | Visual Studio, Javascript, HTML, CSS, React, react-native, react-native-web, Python, Java, Ruby, Rails, Spring, Flask, eclipse, npm, yarn, MySQL, Chrome, git, Atom, NetBeans, PHP  Typically one team is required for development for Linux | Visual Studio, Javascript, HTML, CSS, React, react-native, react-native-web, Python, Java, Ruby, Rails, Spring, Flask, eclipse, npm, yarn, MySQL, Chrome, git, intelli j, NetBeans  Typically one team is required for development for Windows | Mobile apps have different browsers that can be used for testing different web based apps. Mobile app development takes place on other systems, but uses tools like ReactNative, Xcode, Flutter, Android Studio, Java, Swift, Objectiv-C, Firebase, Xamarin  Depending if the development plan is to use cross platform development or native development, multiple teams may be required. |

## 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**:

Because the application will be supporting iOS, I believe that the application should be hosed on a Mac based server. This will allow easy integration using Xcode and the SDK from Apple. All other platforms are easy to incorporate with Mac, but the reverse is not true. I would use React for the front-end with React Native for the mobile application versions. Deploying the browser based React front-end would be very simple and easy with Heroku or similar service with AWS S3 used for image storage as this would be really cheap and simple.

1. **Operating Systems Architectures**:

I would recommend setting up a server-side API using Ruby on Rails due to its easy of use and simplicity run on a Mac OS X server. All images will be stored in an AWS S3 bucket with Heroku which is fully managed allowing the focus to remain on the product without having to worry about maintaining servers and will allow for appropriate scalability.

1. **Storage Management**:

We will use AWS for the bulk of our storage. This will allow for server side image optimization, compressing, and caching which will allow for the game to run faster and take less storage on the users’ devices.

1. **Memory Management**:

The Xcode IDE has a memory leak setting that can be turned on to help manage any memory leaks. In addition, I would recommend caching images as objects locally so that there don’t have to be so many requests to the server for the images to be served up.

1. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>

I think that using a client-server model will allow various clients on different platforms to all make requests using a RESTful API to the server for anything they need to do in the game. I would also recommend using web sockets to make sure the chats and drawings and anything else that needs to be can be in real time.

1. **Security**:

The API in my opinion is the most important part of ensuring that the application is secure.

I think that input validation is incredibly important here. Making sure that only the correct types of input are being received is really important especially in cases of SQL injection or anything else of that nature.

Cryptography is important as all sensitive data should be encrypted when it is being transmitted anywhere. Intercepting data transfer over the internet is fairly easy, so the data itself needs to be protected while in transit.

Code quality is always pretty important in my opinion, but this software would need to accommodate many different types of users, and if the code is written poorly, mistakes on data access can easily occur.

Secure error handling could also be important. There are many cases in which requests to an API fail and need to be handled according to the reason for failure.

Many of these security precautions can be implemented using various libraries that are available on most any operating platform. However, there are some of these that are implemented very well through the AWS API for accessing the data that will be stored there, and Apple offers some of these features through their SDK and some through settings in their Xcode IDE in addition to everything else that would be implemented through other third-party libraries.