

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 | 05/24/2024 | Corbin Badenhorst | Wrote a comprehensive executive summary, listed the design constraints for the project, and described the UML diagram. |
| 1.1 | 06/07/2024 | Corbin Badenhorst | Filled out the Evaluation table with the Development Requirements for each platform. |
| 1.2 | 06/20/2024 | Corbin Badenhorst | Filled out the Recommendations portion of the design template |

**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)

Creative Technology Solutions (CTS) has agreed to help their new client, The Gaming Room, develop a web-based game based on their current game Draw It or Lose It. Currently the game is only on Android, but they would like to be able to branch out to other platforms to widen their audience. They would like the game to have one or more teams, each team to include multiple players, the game and team names need to be unique, and only one instance of the game exist at the same time. In order to meet these requirements, we will create a cross-platform application utilizing modern development practices in order to streamline development and keep feature parity across all platforms. We will make the app fun and easy to use to the users will keep coming back to play.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

1. One design constraint is that the application needs to be developed in a web-based environment. This means the app will need to be accessible over the internet through a web browser, so anyone on any device with a browser can access the app, whether that be a phone, tablet, or computer, on any operating system. Since it will be internet accessible, security needs to be a priority, as we don’t want nefarious individuals accessing users information.
2. Another designed constraint is needing a fluid and easy to use user interface. Having an easy-to-use UI will allow the user to forget about how to use the application and more on how they are having fun. Making it easy to use will bring them back to play more often, compared to a web-app that they must fight with to figure out and navigate.
3. The application will need to be able to scale based on user needs. If lots of people are playing at once, it needs to feel to the end user that only a couple are playing. In other words, the user should have a consistent experience regardless of how many users are logged into the website. This will require optimization of network infrastructure to make sure everything is efficient and works as it should at all times, or as often as possible, baring scheduled maintenance.

The implications of the design constraints will steer us in the right direction of making an app that functions correctly, is easy and fun to use, and efficient to develop and code for.

## [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)

This UML diagram consists of several different classes all having different relationships with each other. The classes include: Entity, Team, Game, Player, and GameService.

The Entity class acts as the base or “main” class that the other classes feed off of. This is part of the inheritance principle in object-oriented programing, since the Entity class provides the ID and name to the to the other classes. Since we are using inheritance here, we can avoid having to type the ID and name under each subclass, since it inherits those attributes from the Entity class, thus saving time and resources.

The Game, Team, and Player classes all directly connect to the Entity class.

* The Game class provides a list of teams participating in the current game and has the ability to add a team to the game, while also representing the current game in the application.
* The Team class list the current players on that team. It also has the ability to add a player to the team using the “addPlayer” method.
* The Player class represents a single player and has a method to retrieve a sting.
* The Game and Team classes have an aggregated relationship with each other, while the Team and Player classes also have an aggregated relationship, since a player can exist on its on, but there would be no team without players.

The GameService class is what starts the whole process of creating a game instance. Since it follows the singleton pattern, only one instance of GameService can be running at a time. This class has the list of current games, and has methods to access different functions, such as adding a game or see who the next player is.

**"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 can be used as a server for web-based apps, but not as popular as Windows or Linux. Based on the Unix environment. Some advantages are since it is based on Unix, it is stable and tight security. Disadvantages are the hardware is limited to what Apple provides and could increase cost. | Linux is a widely known and popular choice in regards to hosting servers. Linux is stable, good security, and sometimes could even be open source. Can be installed on almost any type of hardware, not locked to proprietary offerings like Mac. But it could be more intimidating starting out if you aren’t familiar with the Linux environment compared to Windows or Mac. | Windows might be the most popular choice for hosting web apps, especially if they use Microsoft tech like .NET. Has wide compatibility with different hardware and software akin to Linux and has robust security. It’s not as open source as Linux, as Windows is proprietary software, but not as necessarily closed of as Mac. | Having a mobile device as the host server is not the best idea, as they typically can only be connected to the internet by wifi, which is unstable, and since it is a fixed device, you can’t add memory or storage as your client base grows, so can’t scale well. |
| **Client Side** | Development for the Apple platform consists of using Xcode as the primary IDE. Will need to be fluent in Swift and Objective-C as that is the main programming language for Mac. Costs could be more than Linux or Windows as you need a Mac to use Xcode and Swift isn’t as widely used as other languages since it is just for Apple devices. | Costs for developing on Linux could be considerably cheaper than Mac, as you don’t need proprietary software or computers to code for that platform. Most computers running Linux, Windows, or Mac can program for Linux. | Costs for development can vary depending on the cost of licenses from Microsoft. Most development can be done on Linux, Windows, or even Mac. Expertise in programming languages like C# and .NET will be necessary. A popular IDE for C# would be Visual Studio. | Since they are wanting multiple different mobile platforms, iOS and Android, costs will increase since we now have to support at least two mobile OS’s. Supporting multiple OS’s will take more time, but since they are essentially the same game, we just need to port for compatibility and feature parity. |
| **Development Tools** | Xcode is the primary IDE for Mac when developing in Swift and Objective-C. C++ is also another popular language for Mac. But since the game will be played in a web browser, we just need to be able to code for each systems respective browsers, and whatever is the most popular on the platform. Since coding with Swift requires a Mac and xCode, costs will be more than the other platforms that have free alternatives. | Linux is able to support many different IDE’s and programming languages, not locking itself to just a couple like on Mac. Popular languages are Java and Python. Some popular IDE’s could be Eclipse and IntelliJ IDEA. The development tools all have free options, so development costs can be low. | C#, C++, .NET, Java, and Python will all be good programming languages to know for Windows. Visual Studio is the go to for C# and C++ on Windows for IDE’s, but it also supports other languages too. The development tools all have free options, so development costs can be low. | Android and iOS use Java and Swift as their main coding languages, respectively. Java can be coded in any computer platform, Mac, Linux, or Windows, and Swift is coded on Macs using xCode. Since the game will be accessed through a browser though and not a dedicated app, you won’t necessarily need to code in their proprietary languages, just need to code it to run in the browsers. Now each OS has different default browsers, so we will need to cater to those as well as the most popular 3rd party browsers. Safari, Google Chrome, and Firefox are the big three. |

## 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**: To choose the best operating system to host Draw It or Lose It, we will need something that is scalable, lost cost, and easy to use. My recommendation would be to use Windows Server as the server host platform. Windows is easy to use, can be run on non-proprietary hardware unlike the Mac, and the cost isn’t high. A Linux server could be cheaper, but the support from Windows I feel would make up for the added cost. Windows also has tons of documentation, so if a question comes up with how to fix it, Microsoft has documentation on many different aspects of the operating system you can look up online.
2. **Operating Systems Architectures**: Windows Server uses a multi-layered architecture. Having a multi-layer architecture approach allows the different layers to be separate from each other, providing flexibility between the layers so if one aspect needs to be updated it can be updated on its own without needing to update the other layers. These layers include the data layer, presentation layer, and the application layer.
3. **Storage Management**: Windows Server offers a variety of different options for storage management. You are able to use different storage devices such as hard disks, tapes, and optical disks. It supports using RAID and storage spaces to sort of pool hard drives together to act as one giant hard drive, instead of having to decide which specific hard drive to save something to. It also has a feature called Data Deduplication, where it finds duplicated files on your system and can get rid of them to save storage space and in turn save costs.
4. **Memory Management**: Windows Server offers a variety of different options for memory management. Some of the Windows kernel-mode memory options include memory allocation and deallocation, memory tracking, and file mapping. Memory tracking keeps track of memory locations, while memory allocation and deallocation manages memory virtually and dynamically.
5. **Distributed Systems and Networks**: Since Draw It or Lose It needs to communicate with multiple different platforms all over the world, we would need to incorporate a Wide Area Network (WAN) for the majority of the communications. A Local Area Network (LAN) can be used when local computers need to access the server for maintenance but would not work with all our users since they will be outside the local network. The users’ devices will need to send requests from whatever network they are on, over the internet and to the Draw It or Lose It server, then goes back out over the internet to the users’ devices. The systems would use both Remote Procedure Calls, which allow apps to communicate with each other across a network, and Distributed File Systems, which allow remote access to files stored on the server. This will allow the app to talk and retrieve the necessary data needed to run and display the images. If the user’s internet goes down, the remote server’s internet goes down, or there is a lot of traffic on the network that the server isn’t equipped to handle, it could cause issues with running the app. Would end up making the app run very slow or not at all, since communication would cease if the internet went out.
6. **Security**: Windows Server offers robust security features. These features include data encryption, firewalls, and secure user accounts. Data encryption ensures that even if a nefarious 3rd party gains access to the server, the files stored on them will be encrypted and unusable to the 3rd party without knowing how to access them. Firewalls can help prevent 3rd parties from gaining access in the first place by blocking unknown or unauthorized traffic. Having secure users and passwords ensures access to the server is only done by trusted parties.