

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 | 03/23/2024 | Seth Porter | Added full backend and contributions to this document. |

**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 project, Draw It or Lose It, is a project that is set out to develop a web-based game that should run on multiple platforms. The game should have teams in each game with rounds lasting up to four minutes. A picture will be pulled from a library, and once a team guesses the image, they will score a point, or the round end once time runs out.

## Requirements

The game must run on multiple platforms. Each team should allow up to more than one player and only one game to run at a given time.

## [Design Constraints](#_2et92p0)

The game needs to be able to run on multiple platforms, multiple teams must be present with multiple players making up each team, the game must be able to check for unique team and game names when creating teams, and only ONE instance of a game can be allowed at any given 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)

This UML design operates in the node ideology, meaning that there is one parent node, while the rest of the children inherit from it and add their own data to it. Game, Team, and Player inherit the Entity class. The GameService class is the class that hosts an instance of a game. Each game will have one or more teams in it, each team will have one or more players in it. The ProgramDriver and SingletonTester can be ignored in the final iteration of the project, due to these classes only being used to test the functionality of the game.

**"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** | It’s easy to use with excellent GUI and UX. It also allows terminal commands if needed. This system requires a code signature.  **UPDATE FOR PROJECT TWO:** The first thing to say when it comes to servers on Mac, is server upkeep does cost time and money. Security implementation is about the same in all operating systems when it comes to authenticating WebSocket sessions. Since the client is not asking for heavy security, basic security will suffice, mind you, if the security is secure. In Mac’s case, CORs should be enabled for best practice in Mac’s case.  Rolling out a server for the project would cost a minimal amount when it comes to HTML sockets and an API geared towards clients to connect web browser. This API would be something like Ruby on Rails, Node.JS, Flask, and so on. Only use well known APIs that have be proven by full web stacks. | Free, open source. High learning curve with a lack of third-party support, but easy to develop on. Lacks GUI, but it STRONG in its terminal side.  **UPDATE FOR PROJECT TWO:** The first thing to say when it comes to servers on Linux, is server upkeep does cost time and money. Security implementation is about the same in all operating systems when it comes to authenticating WebSocket sessions. Since the client is not asking for heavy security, basic security will suffice, mind you, if the security is secure. Linux servers operate well with multistep authentication factoring, SSH keygen pairs, and software updates without closing the server completely. Rollbacks in Linux is also a very easy tool to implement. With how easy Linux is to configure, detecting foreign processes in Linux would be easy, so keeping this practice in mind would benefit security.  Linux severs are utilized heavily in cloud services, such as firebase and AWS, so if the client would prefer server hosting to be pushed elsewhere, Linux could provide this. | Cheap, easy to use. GUI based but also allow complex command prompt and PowerShell functions. Very good support with most of the PC base using this OS.  **UPDATE FOR PROJECT TWO:** The first thing to say when it comes to servers on Windows, is server upkeep does cost time and money. Security implementation is about the same in all operating systems when it comes to authenticating WebSocket sessions. Since the client is not asking for heavy security, basic security will suffice, mind you, if the security is secure.  Windows server solutions can begin at a low cost, especially when rolled out with something like NODE.JS or a flask server utilizing the REST framework. Windows and Linux share a lot of similarities when it comes to developing and hosting servers due to the amount of first party and third-party support. With Microsoft constantly providing support to Windows servers, security should be at an all-time high. | This option would not work properly due to the lack of support for coding backend and hosting it on mobile. While this is possible, this option is not viable with the scope given to us because it would require us to develop everything from the ground up, which would ultimately be a very poor financial decision. |
| **Client Side** | Very expensive to use and requires code signature with skills in a niche language called Swift. Easy to navigate though.  **UPDATE FOR PROJECT TWO:** As stated above, developing Macs is usually the most expensive option. The recommended route to take with Macs is hiring an independent developer to develop a hybrid application. This application would have one codebase for multiple platforms. This would be the cheapest route for Mac client development. | A very deep learning curve depending on the distro being used. Cheap to develop in terms of cost and time.  **UPDATE FOR PROJECT TWO:** Since a web API can be used for cross platform support, the best route to take is to develop a single codebase for the front-end client interaction, and front-end JWT-based Authentication can connect to our Flask or Firebase database. The Canvas drawing and chat portion can be streamed over a secured web socket living on the server. React should be considered to make this happen. | Much like Linux but doesn’t require learning new concepts since third party support is massive. Cheap on money and time.  **UPDATE FOR PROJECT TWO:** Much like Linux, a single web codebase could be used to develop in Windows. React would also be a good option. | Very easy to navigate, but it does require more time and money to develop anything for due to the variability in each OS.  **UPDATE FOR PROJECT TWO:** To round off the client side of the project, it would be the best option to use a single web codebase to develop the client due to its adaptability for each OS. React has been mentioned several times and this is due to it being able to run on every listed operating system. This is true for Android and IOS platforms as well. |
| **Development Tools** | As stated before, all code will need to be signed to run on Mac. Mac has full range on most languages, but it mainly supports Swift has its powerhouse language.  **UPDATE FOR PROJECT TWO:** VSCode Homebrew package manager to install Unix and Mac utilities, Xcode IDE, iTerm2 emulator, Tower git client, Dash API browser. Languages and frameworks must also be installed as well. This includes Javascript, HTML, CSS, React, react-native, react-native-web. A SQL style database like mySQL, MariaSQL, PotgreSQL, etc. Flask and NODE.JS would also be required as well. | Linux has first party and third-party support for all languages, including the Cs and Swift. As stated above though, it has a deep learning curve in navigating the OS environment. It also lacks Visual Studio.  **UPDATE FOR PROJECT TWO:** Visual Studio Code, Atom, Vim, bash command line, Git, NODE.JS, flask, Javascript, HTML, CSS, React, React-native-web, NPM, Yard, and a SQL style database like mySQL, MariaSQL, PotgreSQL | Windows supports all languages, besides Swift. There are many tools for app development, including the powerhouse known as Visual Studio. Due to most of the PC base using Windows, many tools are developed with mainly Windows in mind.  **UPDATE FOR PROJECT TWO:** Visual Studio Code, Gvim, bash command line, Git, NODE.JS, flask, Javascript, HTML, CSS, React, React-native-web, NPM, Yard, and a SQL style database like mySQL, MariaSQL, PotgreSQL | IOS only really supports Swift, which takes time and money to learn. APK mainly supports Kotlin and Java. Both platforms support the HTML code family, and this is the preferred route when developing apps for mobile devices.  **UPDATE FOR PROJECT TWO:** Javascript MUST be enabled on all mobile platforms for the game to function properly. |

## 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**: Windows is the preferred OS to develop the project on, due to how much support it has first party and third party wise.

**UPDATE FOR PROJECT THREE:** It is highly recommended to use Windows in the Draw It or Lose It game due to most household computers running the Windows OS. This will provide multiple solutions such as Cordova, React, and Xamarin for the multiplatform aspect of the game. Microsoft also offers a wide range of tools in first party support, along with the internet providing an equal, if not larger array of tools in third party support. Windows also works very well with android systems and it has the most support in web development.

1. **Operating Systems Architectures**: Windows is a GUI based operating system, meaning it is easy to navigate through without needing to know how to use the command prompt or PowerShell.

**UPDATE FOR PROJECT THREE:**  In my research, a hybrid system architecture should be chosen. This design will combine the features of monolithic, microkernel, and layers architectures. Windows NT Kernal is an example of a hybrid system. This system will provide more security out of the three styles listed above. Hybrid systems are also easy to manage due to their layered approach.

1. **Storage Management**: Windows uses something called storage sense to manage its storage devices. This allows the user to manage their files and drives according to how they wish.

**UPDATE FOR PROJECT THREE:** Azure from Microsoft is recommended for this project due to its integration with Windows. The support for Azure is also another reason to use it for storage management along with its pricing. The features that come out of the box with are the Azure Cloud, which allows the use of docker containers, leveraging instances of cloud storage that can be deployed, multiple scaling options for storage, and cost efficiency when maintaining scalability,

1. **Memory Management**: Memory management is done automatically in Windows, but it does allow the user to do so if required, but it is not easy to do it. If the code is set up correctly without any memory leaks, then Windows will do the rest.

**UPDATE FOR PROJECT THREE:**  Windows offers a variety of storage and memory management tools. Azure services and Visual Studio can be used to manage the codebase versions. Memory can be managed using industry best practices to maximize efficiency. With each new edition and update, Microsoft has prove that memory management on the Windows platform should be considered as the go to option.

1. **Distributed Systems and Networks**: A network-based game will most likely include a database to store data on. This will be central sever, while clients will communicate with each other through this sever. Developers must make the servers from scratch because no system exists that works with everything.

**UPDATE FOR PROJECT THREE:**  AS stated above, Azure would be the best option for storage management, but it should be considered a viable option for distributed systems and networks. This is because it offers max uptime using cloud-based system for emails and application insight data logging. It also offers many automations options in keeping up with clients’ needs, which can help cut resource costs, which can help shift more resources into the game itself.

1. **Security**: Windows comes in with built-in security called Windows Firewall. This is as good as most free antivirus software out there. If the user of the OS has basic internet safety skills, then most of the time, the windows security will take of anything that slips past the user.

**UPDATE FOR PROJECT THREE:**  Security threats are a constant in today’s world, so keeping proper security is a must in every application. Using a trusted source of security is the best option in this case. Aura provides protection for Windows, Mac, Linux, IOS, and Android systems. This will come at a cost; this is a necessary one due to ever-evolving threats to security. Aura also offers 24/7 U.S. based customer support, which is a benefit for this service. Another consideration is ensuring company employees understand the importance of password security protocols and implementing an authentication process together with authorization. It also recommended standardizing authorization protocols, that include limiting access to information only when required, versus allowing the same level of access to everyone or allowing everyone access to everything. With the above mentioned in mind and given earlier recommendations of Azure, I recommend the client utilize Azure’s App Service, through their App Service Plan using Active Directory for login. With the added benefit, this service also includes limitations of access per user using in IP configurations, many options in database security, and encryption options for additional protection against security breaches.