

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 | 03/20/2024 | | Tomas Palma | | Executive summary and design constraints generation. |
| 1.0 | | 04/07/2024 | | Tomas Palma | Delve into strengths and weaknesses of different operating systems. | |
| 1.0 | | 04/20/2024 | | Tomas Palma | Make formal recommendation for operating platform, cloud provider, and security measures. | |

## [Executive Summary](#_sbfa50wo7nsh)

After a successful mobile launch of the game, Draw It or Lose It, by the company, The Gaming Room, a transition to incorporate a web-based version of the app has been proposed. The app must serve multiple platforms and be hosted on a distributed system.

## Requirements

* Must successfully operate on both multiple platforms.
* Must be able host the game with multi-person teams.
* Must have the ability to only allow unique game and team names.
* Must be able to handle large amounts of traffic.
* Must have only one instance of the game in memory at any given time.
* Must have a seamless UI/UX that helps retain users.

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

* A decision needs to be made as to which framework the web-app will be used to host the game. I propose using the MERN stack for developing the cross-platform site. This stack is a combination of Mongo dB, Express.js, React, and Node.js to build a fully functional site. In terms of the game itself, I suggest using a low-level programming language, such as C or C++.
* A cloud service must be decided upon to host our cross-platform site. A proper plan must be selected to handle the projected amount of traffic the website is expected to get.
* The cross-platform website should consider which versions of web browsers that it wants to allow compatibility to. Ensuring the security of the user’s data should be a high priority and outdated browsers can lead to data leaks or attacks. Thus, browser version compatibility should be closely inspected.

## [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 structure of the application can be observed by looking at the UML diagram. In it you can see the object-oriented programming principles of inheritance, encapsulation, and abstraction. Inheritance is displayed via the classes Game, Team, and Player, which inherit from their parent class, Entity. Encapsulation, which is the hiding of data, is seen through the use of private variables and constructors denoted by minus signs in the UML. Lastly, abstraction is shown in the diagram through functions such as, addPlayer(), which has a function name that explains what it does but not how it does it.*

*Shown is also the interaction between classes for the program to function properly. The ProgramDriver class which hosts our main method where the application is run from, has an association with and uses the SingletonTester class to ensure that only one instance of game is present within memory at any given time. The rest of the classes, minus the GameService class, are all tied together. The classes being Entity, Game, Team, and Player. The GameService class differs as it is only tied to the Game class and contains the private constructor for the Singleton design pattern to be implemented. Out of this group of classes, there can exist between zero to many at any given times denoted by the “0..\*” symbols.*

**"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 have the benefit of reliable and powerful hardware to host servers for web apps. It also comes with a developer friendly programming system. On top of this, mac technology has strong security features. A downside of Macs on the server side would be its cost, as well as its rigid rules about hardware customization. Apple products tend to be on the pricier end, and they have historically not been fans of customer tinkering with their hardware. | Linux systems are known for their flexibility and affordability. This operating system can be scaled to handle all types of servers. The Linux operating system comes with flexibility in terms of customization also. One real pitfall of the Linux OS lies in its development environment. There is a learning curve with Linux systems. | Windows is a strong option in deciding to host web-apps, in particular .NET framework apps. It also contains a user-friendly interface making the developing and maintaining of servers less arduous. A big downside of Windows technology lies in its security pitfalls. Windows has a history of security breaches which can lead to uneasiness in developers when deciding on which operating platform to select. | When thinking of hosting a server on a mobile device two pros come to mind. The first being portability in how you could host a server from wherever you please. The second is the cost-effective nature of it since phones tend to cost less than a computer. Potential downsides exist in the fact that mobile devices tend to lack in processing power compared to computers. Also, security concerns persist since laptops often have a higher level of security when compared to mobile devices. |
| **Client Side** | Macs are powerful machines which can run programs rapidly leading to a seamless client-side usage of products. This power comes in handy when developing applications. Macs also have an easy interface to interact with. One of the biggest pitfalls of Macs are the price tag. These computers tend to be on the pricier end. | Linux systems are a cost-effective way to develop and host a web-app. On top of this, they also are powerful machines which can run programs quickly with excellent resource management. Again, a real downside of Linux lies in its learning curve to develop in its environment. This can lead to longer development times in more less experienced programmers. | Windows lies in between Mac and Linux in terms of costs. Windows computers aren’t necessarily recognized as hardware juggernauts leading to slower running of applications when using. Windows is not necessarily hard to use and is widely used leading to it being favorable to some on the client side. | When developing for a mobile device there must be a UI/UX geared for smaller screens that is easy to use. Successful apps are typically cross platform, so development be longer than doing it for a single platform. This development will also cost more money since it will take longer and requires a special expertise. |
| **Development Tools** | VSCode is widely popular for its user-friendly interface and endless add on features. Within VSCode for a web app there would be mostly HTML, CSS, and JavaScript to develop the application. In terms of mac specific developer tools there exists XCode which can help developers deploy, develop, and debug programs. Programming languages associated with Mac are Swift and Objective-C | Linux offers VSCode the same as Mac and Windows which has been shown to be a developer favorite recently. Linux also has a powerful terminal which uses tools like yum for package management and general development. Programming languages most associated with Linux systems are C and C++. | Windows offers VSCode for download like other operating platforms. It also has the .NET framework which natively runs on Windows making it a strong option when deciding to develop a .NET app. Programming languages associated with Windows are C# and VB.NET. | A popular cross platform mobile development framework is React Native. This framework uses JavaScript and React to build a mobile app. Another cross-platform programming framework is flutter – which utilizes the programming language Dart. |

## 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**: I recommend using the Linux operating platform to transfer the mobile app into a multiplatform app. Their flexibility, cost-effectiveness, and reliability make it an excellent choice to develop Draw It or Lose It further.
2. **Operating Systems Architectures**: Linux operating systems are open source and operate using monolithic kernels. VSCode can be used for development, as well as implementing yum for package manager. The programming language Java is my recommendation for this development.
3. **Storage Management**: For storage management, I recommend Logical Volume Manager (LVM). It is a Linux favorite and allows for easy scaling, performance optimization, and strong back up memory versions of your code. I also recommend using caching and dynamic loading to ensure the application runs smoothly. Caching deals with storing frequently accessed memory in easy to reach places while dynamic loading deals with loading only content that is needed to decrease memory overhead.
4. **Memory Management**: Linux operates on a kernel system which manages memory. It also features things such as virtual memory and memory protection. Virtual memory can use disk space on top of RAM space if more memory space is necessary. Memory protection helps prevent memory corruption leading to a better user experience. It is also critical that a game that features a high number of image files, such as Draw It or Lose It, stores these images in a compressed state to reduce the amount of memory usage in storage.
5. **Distributed Systems and Networks**: For Draw It or Lose It a cloud services provider should be utilized to host the game servers. I believe AWS to be an excellent choice due to its reliability, security, and scalability. Also, RESTful APIs will also be used to communicate and maintain a cohesive application between different medias. Another component to incorporate are media queries to help the game adjust to different screen sizes which is particularly useful in managing screen sizes between phone, tablets, computers, and monitors.
6. **Security**: To secure user data, robust authentication and authorization methods will be used to prevent attacks. A strong firewall is necessary for keeping unnecessary traffic out. A proper encrypting method must be selected to protect data in transit as well as stored data from being read by attackers. Linux has a longstanding history of strong security standards, which has built in authentication and firewall methods built in. Dependencies for the game must be closely monitored and updated/swapped if found to have a potential weakpoint.