

<Creative Technology Solutions (CTS)>

# **CS 230 Project Software Design**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 10/20/2024 | Francisco Sousa | Draw It or Lose It |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to expand its Android game *Draw It or Lose It* to work on more platforms: Linux, Mac, Windows, and mobile. As a consultant for Creative Technology Solutions (CTS), I’ve evaluated these platforms to help decide which ones are best for hosting the game and making it available to more players. I looked at the pros and cons of each platform, focusing on how they work for both server-side hosting and client-side play.

Requirements

The Gaming Room has asked for the following:

* Expand the Android game to Linux, Mac, Windows, and mobile.
* Host the game as a web-based app that can handle thousands of players.
* Make sure the game works across different platforms, including desktop and mobile web browsers.
* Keep development costs, time, and expertise in mind.
* Ensure the game is secure and scalable.

## [Design Constraints](#_2et92p0)

**Scalability**: The platform must handle thousands of users at the same time.

**Cross-Platform Compatibility**: The game must work well across Linux, Mac, Windows, and mobile devices.

**Cost**: Development and hosting should be affordable, with a preference for cost-saving options where possible.

## [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 UML class diagram shows how the main classes in the application, like User, Game, Drawing, and Guess, are connected. These classes use object-oriented programming ideas like encapsulation, inheritance, and polymorphism. For example, the User class can inherit features from a general Person class, which helps keep the code organized, reusable, and easier to maintain.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is fast but expensive, and hosting options are more limited. | Linux is low-cost, reliable, and widely used for hosting web apps, making it a strong choice for scaling the game. | Windows is easy to use but comes with licensing costs. IIS is good for web hosting, but costs are higher. | Mobile devices can’t host but can connect to the game through a web app. |
| **Client Side** | Mac works well with modern browsers, but development tools are costly. | Linux supports all major web browsers and is free, though it requires more tech knowledge. | Windows is widely used and supports major browsers, but development costs more due to licensing. | Windows is widely used and supports major browsers, but development costs more due to licensing. |
| **Development Tools** | Xcode is used for Mac, but it’s costly and needs Mac hardware. | Linux uses free, open-source tools like Eclipse and VS Code, making it budget-friendly. | Windows uses tools like Visual Studio, which are powerful but need licenses. | Android Studio and Xcode are needed for mobile, and using cross-platform tools like React Native can reduce development time. |

Each platform—Linux, Mac, Windows, and mobile—offers specific strengths and challenges for hosting the game and supporting development. For server-side deployment, Linux is the best option because it is low-cost, reliable, and widely used for hosting web applications, making it ideal for scaling the game efficiently. Mac offers good performance but is more expensive, with limited hosting options. Windows provides easy-to-use hosting through IIS, but licensing fees can drive up costs. Mobile devices cannot host the game but can access it through a web-based app, ensuring compatibility with mobile players.

On the client side, all platforms need to support modern web browsers to deliver smooth gameplay. Mac works well with modern browsers, but development requires Mac hardware and can be expensive. Linux is free and compatible with major browsers, but it requires more technical knowledge. Windows is widely used, easy to navigate, and supports all major browsers, though development costs may rise due to licensing fees for tools like Visual Studio. Mobile development demands expertise in both iOS and Android platforms, adding complexity. However, using cross-platform tools like React Native can streamline the process, reducing development time and effort.

## Recommendations

1. **Operating Platform**: Linux is the best option for hosting *Draw It or Lose It* because it is affordable, reliable, and well-suited for web applications. It is widely used in the tech industry because it can handle high traffic, making it ideal for hosting a game with many users. Since Linux is open source, it also lowers costs by eliminating licensing fees and gives developers more flexibility to customize the platform.

* **Operating System Architecture**: A microservices architecture is the best approach for this project. This means that the game will be built as smaller, independent services, each focused on a specific function, such as managing players or handling game sessions. Microservices allow for easy scaling, meaning individual parts of the game can grow or update without disrupting the entire system. This structure ensures the game runs smoothly, even under heavy usage, and helps with troubleshooting when something goes wrong since each part works independently.
* **Storage Management**: The game will benefit from using cloud storage solutions like Amazon S3 or Google Cloud Storage. These services offer secure, scalable storage that can grow with the game as more users join. Cloud storage also provides backups and redundancy, ensuring that player data and game content are always available, even if something goes wrong. With cloud storage, there is no need for significant infrastructure upgrades, which keeps the system efficient and cost-effective.
* **Memory Management:** Linux offers advanced memory management techniques, such as paging and segmentation, which help the game run efficiently. These methods ensure that only the necessary data is loaded into memory, reducing the load on the system. Additionally, caching strategies will keep frequently used data readily available, improving performance by minimizing delays. These techniques ensure the game responds quickly, even when many users are active at the same time.
* **Distributed Systems and Networks**: Since the game will run across multiple platforms, smooth communication between clients and the server is crucial. RESTful APIs will ensure reliable data exchange between platforms like desktops and mobile devices. RESTful APIs are compatible with different systems, making them perfect for this multi-platform setup. To maintain consistent gameplay, strategies such as retry mechanisms and data synchronization will be implemented to handle potential network outages.
* **Security**: Security is essential for protecting player information and ensuring the game’s trustworthiness. SSL/TLS encryption will secure data transfers between the client and the server. Sensitive data, like player information, will be encrypted both during transmission and when stored. Regular security audits will help identify any vulnerabilities and ensure compliance with privacy regulations. Implementing multi-factor authentication (MFA) will provide additional protection against unauthorized access, safeguarding both player accounts and data.

**References**

Amazon Web Services. (n.d.). *Amazon S3 – Overview*. Amazon. Retrieved from <https://aws.amazon.com/s3/>

Hounsell, D. (2018). *Cross-platform development in C#: A guide to creating games with Unity and .NET Core*. Apress.

Mejia Alvarez, P., Zavaleta, R. J., Ortega Cisneros, S., & Gonzalez, R. E. (2023). *Real-time database systems*. Springer Nature.

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.

Pahl, C., & Lee, J. (2015). Containerization and the future of cloud computing: A research agenda. In *Proceedings of the 2015 IEEE International Conference on Cloud Computing (CLOUD)* (pp. 164–171). IEEE.

Richardson, L., & Ruby, S. (2007). *RESTful web services*. O'Reilly Media.

Stallings, W. (2016). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Wang, Y., & Wu, Z. (2017). Cross-platform development of mobile applications: Frameworks and approaches. *International Journal of Computer Applications, 166*(5), 7–13.