

The Gaming Room

# **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 8**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/23 | Aaron Doss | Added Executive summary, design constraints, and domain model sections. Document Revision History entry added. |
| 2.0 | 11/25/23 | Aaron Doss | Added evaluations |
| 3.0 | 12/8/23 | Aaron Doss | Added server architecture recommendations |
| 3.1 | 12/9/23 | Aaron Doss | Added more server architecture recommendations |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 client The Gaming Room wants us to develop a web-based game for multiple platforms based on their current game Draw It or Lose It. The Gaming Room needs help setting up the environment for this project. Our solution for this is to develop software that meets the clients software and hardware requirements.

## 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)

* Software must work in web-based environment.
* Software must meet clients requirements
* Software must operate similarly to clients already existing Android version

These constraints are important to recognize as they indicate we need to have developers versed in web-based development on the team. It also indicates we should examine the android version to ensure the web-based version operates similarly. Ensuring the software we develop meets the clients requirements ensures the client leaves with a good view of our company to open the door for future business.

## [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 diagram below displays how the “Draw it or Lose it” game should be implemented. As seen there are seven classes to create the environment needed to run the game. The Entity class holds our attributes of id and name that its subclasses Game, Team, and Player will inherit. Player extends Entity to hold information regarding the player in the game and outputs that information when necessary. Player and Team are related to one another with a 0…\* relationship indicating a team can have none to many players assigned to it. Similar to this relationship, Game and Team have a zero-many relationship. This indicates that a game can have zero to many teams in it. The Team class holds a list of players that can be scanned to check for matching records. The game class holds a list of teams that can be scanned in a similar way. The GameService class has a zero-many relationship with the Game class. This indicates a GameService can have zero to many ongoing games. The GameService class holds a list of games. GameService is also a singleton class to ensure only one instance of a game to exist at any given time. To ensure this happens there is a singletonTester class which is utilized by the ProgramDriver class.

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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 built in web-server commands to configure and access servers. Mac users report ease of use and relatively low costs associated with server side development. Mac servers run on limited hardware configurations which can be problematic. | Linux offers open source software built around the linux kernel. Linux servers offer stability, security, and flexibility above other environments. Linux has a higher learning curve for entry, however, this can be worked around. Amazon offers online linux services at around the same price point as other environments. Third party sites offer linux server hosting at up to $100/mo. Users estimate building an in-house server can cost around $500 - $1000 for a basic server. | Through the use of VMs windows can virtualize other OS environments. Users state windows has ease of use, web extensions and SQL support. Microsoft offers server solutions through Microsoft Azure at relatively low costs. Amazon offers similar services at low costs through windows. | Using mobile devices to host a server isn’t as viable as other OS’s. This option is not recommended with what we need to adequately solve the clients issue. If we were to use mobile devices it would require a ground up approach for coding, hosting the server and cross-app development. |
| **Client Side** | Macs are generally a more expensive system in comparison to Linux and Windows. Ease of use is comparable to windows, however, transitioning from windows/linux environment to macOS environment can take time to adjust. | Linux is the standard for software and web development due to its open source programs that work in sync with the system. Linux systems are generally the cheapest option, however, this comes at the cost of Linux systems requiring a greater amount of learning time to use them at their full capacity in comparison to windows and mac. Linux devices also offer the most secure development environments. | Windows offers several unique tools for virtualization of other environments. Windows has extensive support for development through Microsoft. | Mobile offers the convenience of accessing the app at any time in any location. Several screen size possibilities to be accounted for. Mobile apps have an intuitive interface design due to their smaller size. |
| **Development Tools** | VSCode lightweight code editor, iTerm2: open-source terminal emulator, Homebrew: open-source software package management system,  Git client, Dash api. Languages: Javascript, HTML, CSS, React, react-native-web, react-native, SQL/mySQL, node.js.  Chrome development tools also available. | VScode, Atom, Bash command line, git, node, flask, Vim  Languages:  Javascript, HTML, CSS, npm, yarn, react, react-native, react-native-web, mySQL. Chrome development tools available, Amazon AWS available | VScode, Gvim, git, Git bash, node, npm, yarn, Microsoft developer tools, .NET Framework, Microsoft PowerToys, Azure.  Languages: HTML, Javascript, CSS, React, react-native, react-native-web | Browsers: Firefox, OperaGX, Chrome, Samsung internet app. Website should be accessible across all browsers and the app. App: Javascript to allow app access, should be accessible through playstore, apple app store, and other app stores. |

## 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**:

For our initial purposes I believe setting up Infrastructure as a Service or IaaS would be a good start. It would be cheaper initially to outsource all the server hardware requirements rather than building up our own server. Ensuring the service is a metered service will guarantee The Gaming Room is only paying for what they use thanks to rapid elasticity. As only the infrastructure aspects are being outsourced, all software requirements will still be in the hands of The Gaming Room. All maintenance of the infrastructure will be handled by the outsourced party. Regarding the OS, I would suggest using Linux OS for its low cost and following reasons. Linux offers improved security versus malware and cyber threats. Linux also allows for more freedom due to its open-source nature. IaaS cloud solutions are compatible with Linux so there should be no issues running a Linux server through a popular service like Azure or AWS that offers a great deal of support on their end. Eventually if the game grows enough and profits allow for it, The Gaming Room can move to setting up their own data center with their own servers. This is, however, not initially necessary due to the smaller scope of the game and large costs associated with building a data center.

1. **Operating Systems Architectures**:

Linux is a modular architecture which allows users to customize and configure their system to suit their needs as stated previously Linux is designed to be scalable and versatile making it an ideal candidate for cloud infrastructures. The architecture of a Linux system is layered structure with several components. This includes: hardware, the Kernal, the Shell, and your Applications and Utilities. The Kernal being the central component of the Linux OS. The kernel manages system resources and provides service to applications.

1. **Storage Management**:

As we will be using a serverless architecture, storage management will be handled by a third party based on The Gaming Room’s needs. Within Linux there are several different file systems that can be utilized, but the XFS file system is the current default in RHEL 7. XFS is a high performance system designed to be highly scalable which suits our purposes perfectly. RHEL 7 supports backup and restore utilities specific to XFS as well, which ensures the safety of The Gaming Rooms data assuming there were a drive failure. Assuming we were to scale the storage purposes up RHEL 7 offers information for setting up RAID arrays for increasing the redundancy and performance of any future expansions if The Gaming Room decided to set up their own data center.

1. **Memory Management**:

In regards to memory management, Linux uses partial memory execution through the use of VRAM. This keeps physical memory limits to a minimum while ensuring load and run times for the app remain quick. This is important as the CPU only needs to run the app as it exists partially in memory. Linux utilizes multi-gen LRU an alternative LRU implementation used for improving performances under greater memory constraints. This implementation ensures that whenever a page fault occurs the least recently used page is replaced with a new one. This implementation gives fewer page faults than other algorithms and allows for complete analysis of the process. The only downside is that the algorithm itself is complicated and will require employees who work with it to be knowledgeable of how it works.

1. **Distributed Systems and Networks**:

The goal of any application is generally to see an increase in use from the target audience. As our initial plan is to employ serverless architecture that will let us scale our servers size to meet user growth, this should be an achievable goal. As the application grows in popularity, the application will have to run across several servers. To ensure that one server doesn’t get overloaded, the use of load balancers will have to be implemented to distribute traffic across the servers equally. This will improve response times and availability of the server, which will in turn result in improved user experience. In distributed systems, the applications database is run across several devices in unison. These databases must be synced constantly to ensure they appear as one instance. The servers should also be set up with redundant power supplies to ensure that if a PSU failure were to occur the server wouldn’t experience any downtime as another PSU would take over. Ensuring that any storage within the server is hot-swappable will also ensure that the server experiences a minimum amount of downtime as any drive failures that were to occur could be quickly replaced without any scheduled downtime. Overall, regarding the benefits of using a distributed system will ensure that the server does not have a single point of failure. This increases fault tolerance and the reliability of the server. Distributed architectures also assist with scalability and flexibility needs as well as overall speed performance boosts for the server. The primary downside to this architecture is the cost. Assuming the Gaming Room were to eventually create their own server to host the game, the creation of a distributed system although beneficial will come at a large cost. The benefits of using one, however, outweigh the negatives.

1. **Security**:

Security is an extremely important feature of a servers architecture. In order to ensure The Gaming Room’s users have their information protected, we will implement AAA into the server. AAA stands for Authentication, Authorization, and Accounting. For the Gaming Rooms purposes the first two A’s are most important. Ensuring that anyone accessing the server has authenticated their identity with a username and password or if the Gaming Room wanted to bolster security more through the implementation of Two-Factor authentication. These will ensure that only individuals who know those authenticating items can access their accounts. Authorization is also important to ensure that when a user logs in, they only have access to parts of the server they are meant to have access to. For most users that will be the front end, only seeing what the developers and administrators want them to see. Accounting is important as well. Assuming a breach were to occur, we will want to know what the intruder did on the server while they had access. This will help implement future preventative measures for any security breaches that the Gaming Room was unaware of. This model can be distributed across all platforms so that anyone who logs in on any device/platform has to go through the same process of authenticating their identity and the server authorizing their permissions based on their role on the server. Aside from this ensuring that web traffic occurs through SSL/TLS certificates will ensure that the client and server data being transmitted is encrypted and secure. The Gaming Room should ensure that their servers are being protected by a firewall. These measures should ensure that user data is protected alongside the overall server.

Citations:

*Distributed Systems explained*. Splunk. (2023, August 29). https://www.splunk.com/en\_us/blog/learn/distributed-systems.html#:~:text=Distributed%20systems%20reduce%20the%20risks,nodes%20without%20severely%20impacting%20performance.

*What is AAA in networking?*. NetworkRADIUS. (2021, February 10). https://networkradius.com/articles/2021/02/10/what-is-AAA-in-networking.html

Mahajan, U. (2022, August 5). *LRU page replacement algorithm*. Scaler Topics. https://www.scaler.com/topics/lru-page-replacement-algorithm/

Coombs, K. (2019, March 19). *Storing data in a serverless application*. OCLC. https://www.oclc.org/developer/news/2019/storing-data-in-a-serverless-application.en.html

*Chapter 3. the XFS file system red hat enterprise linux 7*. Red Hat Customer Portal. (n.d.). https://access.redhat.com/documentation/en-us/red\_hat\_enterprise\_linux/7/html/storage\_administration\_guide/ch-xfs

Mitra, N. (2023, April 18). *Linux architecture*. Scaler Topics. https://www.scaler.com/topics/linux-architecture/