

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 3.0

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 08/17/25 | Logan McCullough | Adding recommendations |

.[Executive Summary](#_sbfa50wo7nsh)

Draw it or lose it, the game loosely based on classics like Win, Lose or Draw. The game will involve teams waging against each other to guess the puzzles as it's slowly revealed. There will be four rounds that are timed, with chances for teams to answer during and after the draw phase. For this to work it must have a fully functional, with room for growth, web-based app. The main system will handle managing the rounds, teams, and players, making sure that one game exists at any time and every name is unique; players, game, team, etc. This will have to be flexible, as there may be multiple teams in each game, multiple games, and players in each round. We will also have to calculate the hardware to judge scalability and future proof. This document will outline the constraints we foresee. It will manage the game, force unique naming conventions, and reliably make everything work in this distributed environment.

## Requirements

***Functional:***

*Game management*

* *one instance at a time (singleton)*
* *create games with unique name*
* *4 rounds 60 seconds*

*Team management*

* *Users can create and join teams*
* *Team must be unique name*
* *Teams have multiple players*
* *Players assigned to one team per game*

*Guesses*

* *Real time drawing rendering*
* *Submit guess during draw phase*
* *One guess for other teams 15 sec limit*

*Name Validation*

* *Games, team, player, check and block if already existing*

***Nonfunctional:***

*Response times*

* *Multiple users interacting time sensitive guesses and times rounds*

*Scaling*

* *Multiple users for multiple teams*

*Cross platform*

* *All modern web browsers*

*Reusability/Modular*

* *High documentations*
* *Expandable and swap pieces easily (potential for themes)*

## [Design Constraints](#_2et92p0)

Web based – It all must be able to be used in browsers with no external addons.

Distributed model – Must be able to be access from multiple people from different areas

Unique names – The system would need to check all name in the list and make sure no duplicates, stopping from being created

Single game at a time – One game instance is going to be active at any given time

Cloud – Must be cloud available for updates S

## [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)

We have a few pieces or classes like, Entity, Game, Team, Player, GameService, SingletonTester, and ProgramDriver. This is the main building block for the entities; team, game and players. This proves inheritance, as those will be passed to each of those classes to make sure there are no duplicates. It has private attributes of id and name and basic methods like getId, getName, toString. With the symbols0...\* Each game can contain multiple teams, which can contain multiple players, which shows a composition relationship. The GameService Class acts like a singleton, letting only one game instance exist at a time. The SingletonTester and ProgramDriver are used to verify singleton behavior. This is going to be more modular to allow transformation or additions in the future as each class is only accountable for what is needs to run and then passes it on as needed. Making it more flexable for future issues or concerns.

**"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 is not the usual go-to for web servers. It can run Xcode to build and develop iOS apps which would help build the mobile app if we allow it. It can also be integrated with Apache for testing. It does get expensive on the hardware side and some of the MacOS would limit the scalable side. Can use the mac for developer workstations but not for server side. | This would likely be the best OS for web servers. It's free, and already works well with cloud platforms like AWS, Apache and NGINX that are easy to setup. It is less demanding on the hardware so less hardware costs. The downside is (unless running Ubuntu desktop which adds to demand even slightly) its more command line based so would require pure coding. | Windows would be second on the list here especially for this hosting. It has good built in tools like PowerShell for automation, it supports hosting like Apache and NGINX. Docker for Linux based stacks for the flexibility of Linux but windows machine. The downside is windows requires licensing that can add up to be quite costly. | This is not built for hosting or running a server. It would still need one of the other three. Battery, processor limits and memory on phones makes it less than ideal/impossible. OS and app store restrictions would teeter on violating TOS. |
| **Client Side** | Like the server-side notes, it would be great/required to have a developer device with Xcode on apple hardware. The apple developer program has almost a 100$ a year cost to host on the Appstore. We would need developers with a knowledge of Apple tools, like Swift, iOS and Appstore guidelines. It would also give them compatibility testing with Safari if web browser based. | It would mainly involve making sure the client would work well with browsers like chrome or Firefox, no need for a costly Linux based app. The developer tools are all free and open source so again no fees. The development would be quick if using normal HTML, CSS, etc. Testing on Linux though would probably need someone familiar with the environment, to make sure it is able to troubleshoot. | Developing for this is going to be lower cost when doing web-based apps and pretty simple. Since it will go through browser like chrome edge or Firefox, and using normal HTML CSS etc. Since it would have no cost to support windows. Windows wouldn’t require extra workflows as is already expected. If we go to apps, we will need more time like c#, or visual studios. | This will need more cost if we go to the two main android and iOS. For normal web based off the mobile it would use the same HTML CSS and such which would keep the cost and time sink low. App building specifically would need specialized expertise in Swift and Java for IOS and android respectively. IOS would need the developer account with a 99$ yearly fee but android would be free in android studio. Now each app would need more time, and development would be slower but could use cross platform tools to help speed it up. |
| **Development Tools** | If we go for MAC or iOS, we will need Swift with Xcode which would be the IDE for apple. If we do browser based, we will need standard HTML CSS and JavaScript. | A little more freedom with this one. We can use things like JavaScript, Java, Python and C++. Same web-based HTML CSS and Node.js. The IDE’s we could use are Eclipse, and PyCharm. Docker, and Git would also be good tools. Which are all free. | Visual studios and VS Code are popular IDEs. It comes with developer tools like C#, JavaScript, HTML, Python Java. Git and Node.js are good for backend support. Visual studio has some costs for certain versions but free for community needs. | Swift for iOS or Java for android would be great. The tools themselves are free, but they do come at a cost to publish them of 25 dollars one-time fee for android or a yearly 99 for apple. Some tools like VS Code or emulators for testing would be needed. |

## 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**: There would be two recommendations, and it would depend on the comfortability of developers in which development. Initial recommendation would be Linux hosted on AWS. It is wonderful with multiple platforms we are expecting like windows, mac, android, and iOS. Linux is open source with no license fee, AWS gives EC2 and S3 which easily can scale if needed. Performance-wise it has amazing stability. Second would be linux on Google cloud platform for similar reasons. It has firebase and firestore to sync and save, and solid SDK to make sure the mobile players are easily suited. It has a bunch of built in enacting tools like cloud build and cloud deploy.

(N.d.). Retrieved from <https://aws.amazon.com/blogs/gametech/leverage-fully-managed-containers-to-host-multiplayer-games-at-global-scale-on-amazon-gamelift/>

(N.d.-a). Retrieved from <https://d1.awsstatic.com/whitepapers/optimizing-multiplayer-game-server-performance-on-aws.pdf>

1. **Operating Systems Architectures**: With either of the operating platforms recommendations I would recommend a “Hybrid Kernal Architecture” giving it the best of both worlds from monolithic and microkernel. It allows dynamically loadable modules giving the ability to change or move features without altering the original core, letting it scale comfortably with peak performance. The linux kernel has built in security like AppArmor which helps secure data. It also is heavily supportive of containerization with Docker to help grow and expand server resources during peak time and scaling back during slower times.

Linux kernel. (2025). Retrieved from <https://en.wikipedia.org/wiki/Linux_kernel>

Whittaker, G., & George Whittaker is the editor of Linux Journal. (n.d.). #linux. Retrieved from <https://www.linuxjournal.com/content/best-practices-and-strategic-insights-dockerizing-your-linux-applications>

1. **Storage Management**: If we go with AWS and Linux, I would recommend we go with RDS and S3 as combo storage. Main game like users matches leaderboards etc, would be on the relational database storage. S3 would be more for asset storage like images, drawings, any sounds we add, data logs, backup etc. RDS would give great constant and consistent logic for game information and S3 would be great for larger assets.
2. **Memory Management**: Linux with AWS uses quite a few different management features. It does a virtual memory concurrent to physical ram, that way during heavy use or spikes so it doesn't crash with the allocated separate pieces. Demand paging, making sure only what is needed or shortly suspected needing is setup to be loaded and used making scalability and resource support easier. Along with the virtual memory it uses caching to allow frequently used code to be allocated there instead of having to read and write to disk for it each time. Linux kernel also has ‘slab allocation’ which gives short term create and end systems like specific objects or a game session drop off points to reduce fragmenting. Linux also has OOM Killer to help reduce non essentials tasks and processes to keep essentials running.

Linux out of Memory Killer - Knowledge base. (n.d.). Retrieved from <https://neo4j.com/developer/kb/linux-out-of-memory-killer>

1. **Distributed Systems and Networks**: I recommend doing a distributed client-server architecture hosted on AWS. REST API’s for users infio like profiles and leaderboards, and WebSockets for gameplay. Its the low latency combo that drives performance. As stated prior have containers like Docker (or Kubernetes) to keep scaling during peak hours. Databases themselves split by long and short data. Something like RDS for game data, and Redis for the sessions itself data. AWS has a load balancer to ensure traffic is routed properly AWS ELB. CloudFront or CDN to make sure it's routed quickly. With the outages or intermittencies we would use session tokens, which any drops or stops in connectivity can rejoin the game. AWS SQS keep components separate so one disconnect or session failure doesnt send all of them failing.

MozDevNet. (n.d.). The Websocket API (WebSockets) - web apis: MDN. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>

Learn kubernetes basics. (2025). Retrieved from <https://kubernetes.io/docs/tutorials/kubernetes-basics/>

(N.d.-a). Retrieved from <https://aws.amazon.com/elasticloadbalancing/>

1. **Security**: Any communication between the players and the server would use TLS/SSL (since we said we are recommending REST earlier HTTPS) to amke sure malicious intervention or interception is handled. User account information would be encrypted with AWS Key Management System since we are recommending AWS. We would use rolebased access control or AWS IAM to make sure the users only have the minimum amount of permissions needed to access or play depending on their role. We would make sure to use an authentication system like JWT tokens for login and session data. All these recommendations adn concerns while making sure to follow mandated security like the CCPA for data handling behind the scenes.

Cloud compliance and regulations resources. (n.d.). Retrieved from <https://cloud.google.com/security/compliance>

(N.d.-a). Retrieved from <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>

(N.d.-a). Retrieved from <https://aws.amazon.com/kms/>