

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View**](#_heading=h.4d34og8) **4-6**

[**Domain Model**](#_heading=h.2s8eyo1) **6**

[**Evaluation**](#_heading=h.17dp8vu) **7-9**

[**Recommendations**](#_heading=h.26in1rg) **10**

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Document Updates |
| --- | --- | --- | --- |
| 1.0 | 01/23/24 | Conor Steward | Written executive summary. Documenting business and technical requirements for the software. Outline design constraints for the software. UML model description. Development requirements for mac, windows, linux, and mobile device systems. Recommendations for operating platform, operating system architecture, storage management, memory management, distributed system and network, and security. |

## [Executive Summary](#_heading=h.35nkun2)

This exciting new game will need the support of a system that will hold teams, players, and the games and can output them for reference as the game goes on. This support system will also be able to check against existing teams, players, and games to ensure that duplicates cannot be made. There can also be only one game going on at a time. When creating the architecture that allows the game to function with user inputs for guessing each player and team has a unique name and id that can be referenced as needed.

## Requirements

This gaming software will have the ability to have one or more teams involved. Each team will have multiple players assigned to it. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name. Only one instance of the game can exist in memory at any given time. To solve each of these requirements respectively the software will have a team class and player class with unique identifiers. Each team and player will be the only instance of that team or player to exist, which will be solved with iteration. Game class can be made to only exist as a single instance by making it a singleton also with a unique identifier.

## [Design Constraints](#_heading=h.1ksv4uv)

*Network Latency:* The time it takes for data to travel from the client to the server can introduce delays. The best ways to deal with these is client-side prediction, interpolation, and lag compensation can be employed to deal with latency.

*Bandwidth Limitations*: When developing web-side applications, the bandwidth of the users will affect the playability of the game. To address this asset compression, data serialization, and content delivery networks(CDNs) can be used to help mitigate the effects of slow bandwidth.

*Browser Compatibility:* To ensure that the game can run on a variety of web browsers, testing must be done on different browsers. In addition, building the software with HTML5 can ensure that compatibility will be likely.

*Security Concerns:* Encrypting sensitive user/company data, implementing secure communications protocols, and writing the web side so that common security threats such as cross site scripting or cross-site request forging are prevented.

*Device Diversity:* The large array of devices that are used to play web-based games means that the game interface should be adaptable to fit different screen sizes. Also, performance optimization should be considered so the game may run well on a variety of operating systems.

*Statelessness of HTTPS:* The stateless nature of HTTPS, the underlying web communications, means that cookies or tokens must be used to preserve game state, user progress, and authentication information.

*Sandbox Environments:* Direct access to hardware or file systems may not be possible from a web based application due to security protocol.

*User Experience:* Modern standards of gaming set a high bar for user experience. To keep up with this demand, optimized graphics, interactions, and animations are vital.

*Backend Scalability:* Ensuring the backend is built so that the game can support an adequate number of games/users as one time is essential for a smooth gaming experience. To support this, cloud services, catching, and load balancing can be used to handle demand.

## 

## [System Architecture View](#_heading=h.44sinio)

Operating Platform:

For your game Draw It or Lose It, you should work with various Linux servers. The application is running well and the goal from here is to make it cross-platform compatible in the best way possible. Linux offers Apache web servers, Node.js application servers, and can run MySQL as a database server. Node.js and MySQL are particularly useful in their cross platform compatibility and Apache web server can be utilized from anywhere you have an internet connection.

Operating System Architecture:

Apache will serve as the front-facing web server, handling HTTP requests from clients and forwarding them to the Node.js application server. Apache can be configured to act as a reverse proxy, passing requests to the Node.js application server through various proxy modules. Implementing security measures, such as SSL/TLS for encrypted communication (HTTPS), and configure access controls to restrict unauthorized access will be vital.

Node.js acts as the application server, running server-side JavaScript to handle dynamic content generation and application logic. Node.js applications can be deployed using a process manager like PM2 or systemd. These tools ensure the Node.js application stays running and can be easily managed. Leverage Node.js's event-driven, non-blocking architecture to handle concurrent connections efficiently. Use frameworks like Express.js for middleware management.

MySQL serves as the relational database management system for storing and retrieving game-related data. Node.js interacts with MySQL using a driver. We will want to establish a secure and efficient connection pool for managing database connections. Design a suitable database schema based on the game's data requirements, ensuring proper indexing and normalization for optimal performance.

Clients interact with the Apache web server, which acts as a reverse proxy. Apache forwards dynamic content requests to the Node.js application server. Node.js processes requests, performs game logic, and communicates with the MySQL database for data retrieval and storage. Responses from Node.js are then sent back through Apache to the clients. We will need to implement load balancing for the Node.js application server to distribute incoming requests efficiently across multiple instances. In addition, we will use caching mechanisms for static assets and frequently accessed data to reduce the load on the database and improve overall system performance. Implementing monitoring tools to keep track of server performance, resource utilization, and potential issues will prove useful in maintaining Draw It or Lose It. Finally, configuring logging for both Apache and Node.js to capture relevant information for debugging and security analysis to keep the system running properly and safely.

Storage Management:

Utilize a CDN to distribute and cache static assets faster to users, improving load times and reducing the workload on the servers. Integrate Multer, a Node.js middleware, for handling file uploads as it works seamlessly with Node.js and is often used with Express.js. If we use Redis as an in-memory data store for session management it enhances performance by allowing quick access to session-related data. Use a software for caching frequently accessed data to help reduce the load on the database by storing frequently retrieved data in memory. We should implement a robust backup for MySQL, including regular automated backups and offsite storage of backup files. Enable encryption for stored data to enhance security and utilize SSL/TLS for secure communication between the web server, application server, and the database. Develop data archiving and cleanup policies to manage outdated or obsolete data. This helps maintain efficient storage utilization over time.

Memory Management:

The Linux kernel utilizes a sophisticated memory management system that includes virtual memory, page caching, and memory allocation strategies. Virtual memory allows the operating system to use disk space as an extension of RAM, providing more space for applications. Page caching helps in optimizing file system performance by caching frequently used data in memory. Apache uses a process-based model for handling client requests which means that each client request is served by a separate process or thread. Apache's memory management involves allocating memory for each process or thread to handle incoming connections. Node.js utilizes an event-driven, non-blocking I/O architecture, which allows it to handle concurrent connections efficiently with a single-threaded event loop. This event loop is responsible for managing asynchronous events, and the memory management focuses on allocating memory for event handling and maintaining a low memory footprint. MySQL's memory management is crucial for the performance of database operations. Proper sizing and management of the buffer pool are essential for optimizing database performance. Node.js uses automatic garbage collection to reclaim memory utilized by objects that are no longer in use. Understanding and tuning garbage collection parameters can help optimize memory management.

Distributed System and Network:

For Draw It or Lose It a well designed distributed system and network are essential to the continual function, smooth operation, and maintainable state of the game. Below are outlined solutions to avoid various issues. First, break down the functionality of Draw It or Lose It into microservices or services, each responsible for specific tasks. Each microservice can be developed and deployed independently, enabling flexibility and scalability. Designing clear and well-documented APIs to facilitate communication between different components and platforms is vital for user authentication, game state synchronization, file uploads, and any other interactions required between platforms. Message brokers help decoupled components, ensuring that even if one component is temporarily unavailable, messages can be processed later. Ensure that communication protocols are standardized to work across different operating platforms. Robust error-handling mechanisms will also be necessary to address connectivity issues or outages. Use retry mechanisms for failed communication attempts and implement timeouts to prevent indefinite waits. In addition, we should employ load balancing mechanisms to distribute incoming requests across multiple servers or instances, ensuring better resource utilization and scalability. There will need to be clearly define dependencies between components and platforms. Implement secure communication using HTTPS for web-based interactions and secure channels for other types of communication. Additionally, authentication and authorization requests ensure that only authorized parties can access and modify sensitive data. Encrypt data in transit using protocols like TLS/SSL to protect sensitive information during communication over the network. Lastly, we will utilize real-time monitoring tools to detect anomalies, performance bottlenecks, or connectivity issues in the distributed system.

Security:

A robust security system and advanced protocols will be vital for protecting user and company data in Draw It or Lose It. To start, we will implement the use of HTTPS (TLS/SSL) for secure communication between clients and servers. With Apache, we will use TLS by obtaining and installing an SSL certificate. This encrypts data in transit, preventing unauthorized access to sensitive information during communication. Implementing strong and secure authentication mechanisms will be essential for security. Also, we will use secure password hashing algorithms to protect user logins. Consider multi-factor authentication for an additional layer of security on top of these. Best practice is to define clear authorization controls to ensure that users can only access the resources and features appropriate for their roles. Designing secure session management to protect user sessions from attacks will work well. As previously mentioned, we will utilize MySQL's encryption features to encrypt sensitive data at rest. Use transparent data encryption or other encryption mechanisms. Salting should be employed to prevent precomputed attacks on hashed passwords. Middleware in your Node.js application to validate and sanitize user inputs will be essential for data security. Protect against common vulnerabilities like SQL injection, cross-site scripting, and cross-site request forgery. Set appropriate security headers in Apache to enhance the security posture of the web application. Headers like Content Security Policy (CSP) and Strict Transport Security (HSTS) can be utilized for this. Utilize real-time monitoring tools to detect anomalies and potential security breaches and set up alerts for unusual activities or patterns. We will develop an incident response plan outlining procedures to follow in the event of a security incident and train the team on how to respond to security incidents promptly. Also, we will educate users about security best practices, such as creating strong passwords and being cautious with personal information.

## [Domain Model](#_heading=h.2jxsxqh)

The Entity Class is the superclass for the entire model below. This means that Team, Player, and Game Classes all inherit methods from the Entity Class. To access the variables in the Entity superclass the get() methods may be called. The way that Entity is designed, as a singleton, only one instance of the Entity can exist at once. This single instance would serve as a global point of access for the subclasses. The games driver ProgramDriver uses the SingletonTester to test the singleton state of the Entity Class. Designing the classes so that they are modular keeps maintenance/updating/refactoring more simplistic. Also, running the main() method from its own file ensures that code is kept legible and easier to read. The Game Class and Team Class are related to one another with a 0-to-many multiplicity meaning that 0 or more of one class can be associated with 0 or more of the other. The same relationship exists between Team and Player also. GameService Class is also a singleton. This ensures that only a single instance of game can exist at once. The software requirements will be met by the multiplicity of these relationships but also the singleton nature of the Entity Class.

**"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](#_heading=h.z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Xcode and a robust number of server-side tools make server side development easier. Expensive by comparison to other development options, due to the need for cloud based services which incur costs on an hourly basis. Expertise is harder to find in mac development, but not necessarily more expensive if using cloud based servers. | Due to Unix’s table, secure, cost-effective, deep documentation, resource efficiency, package manageability, and scalability are all advantages due to Linux's open-sourced nature, variety of distributions, and security features. Hardware compatibility, fragmentation, and desktop-centric development are disadvantages of these same characteristics. No licensing fee makes server management easier with Lunix/Unix. Server costs can run from $800-$3k as need for high RAM and SSD increase. | Support from Microsoft, easy development, and wide hosting capabilities are due to Windows parent company Microsoft's involvement, simple GUI, and wide software compatibility. Due to Microsoft Licensing costs are high, even if buying your own servers. Server cost itself would be as much as a few hundred dollars to thousands as your needs scale. Not as resource efficient as linux, which would require more resources for the same functions. Limited open source compatibility. | Cloud integration, load balancing, serverless architecture, and robust catching strategies are advantages due to mobile developments variety of backend frameworks, good scalability, API support, and semi-constant internet connection. Costs are similar to mac development since the applications servers would be cloud based. Resource intensive downloading can hamper an applications performance. Mobile devices are susceptible to security weaknesses. Platform diversity can also make development difficult. |
| **Client Side** | Ecosystem integration is seamless, security makes for a safe browsing environment, and performance is high with mac hardware. Browser compatibility is low from safari focused applications and mac hardware is expensive by comparison. Building the application with platform agnostic languages will be necessary for all of the operating platforms. | Open-sourced nature, many software languages, and deep community support are available due to Linux's wide range of development libraries, command line interfaces, and open sourced existence. This same diversity leads to fragmentation of UI. Limited software support due to its open sourcing. Not good for graphics heavy development. Inexpensive compared to other options. | Windows development licenses are expensive. Windows server costs are high compared to other options. Development tools like Visual Studio are time efficient. Cross-compatibility with different windows versions is costly and time consuming. These IDEs, cross-compatibility concerns, and less-tight security require expertise in the Microsoft environment but due to a large market share these individuals are widely available. | Diversity of mobile devices can lead to expensive and time consuming testing. Platform specific needs require that multiple iterations of the application exist in each language. Maintaining these multiple iterations will be more expensive than cross-platform solutions. The diversity and multiple applications means development and testing will take longer and be more expensive. Knowledge for coding in multiple languages, cross-platform tools, and device specific features means finding the right team can be challenging. |
| **Development Tools** | Xcode is well developed and easy to use. Swift, Apple's programming language, is a great tool but is also expensive to develop in. Lower market share means that developers who know how to develop on Mac are fewer. | C++, C#, python, and Rust can be used. Unreal engine, Unity, and Godot Engine can be used as engines. Visual studio, CLion, and Mono Development are sufficient IDEs for linux. Git can be used for version control. Lots of developers can write in these languages and IDEs meaning they will be less expensive. | C#, C++, Visual Basic, and Java can be written in Visual Studio or in Java Development Kit. MySQL and Entity can be used for server side development. Git can also be used for version control on windows units. Developers for windows are common due to market share and should be less expensive. | Swift, Objective-C, Java, and Kotlin languages can be written in XCode and Android Studio respectively. Cross-platform development can be utilized using Flutter or React Native. Github can be used for version control. Knowledge for coding in multiple languages, cross-platform tools, and device specific features means finding the right team can be challenging. |

## 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**: The best operating platform for Draw It or Lose It for expanding to other computing environments would be linux. Its customization and open-sourced nature mean that development is less expensive, fast, and has large community support.
2. **Operating Systems Architectures**: Linux uses a Kernel to handle software/hardware communication and handling hardware resources. Linux provides system libraries that the kernel uses as a bridge to the application. Device Drivers facilitate communication between the kernel and hardware. Shell is utilized as the command line for Linux and is used to interact with the operating system.Linux can function as a firewall, server, and router for the application as well.
3. **Storage Management**: For storage management Ceph is a good solution. It provides object storage, block storage, and file management that can handle large scale use. It is also horizontally scalable allowing for many users gaming at once.
4. **Memory Management**: Linux utilizes virtual memory, giving the appearance that each process has its own address space. This allows multiple processes to run simultaneously without interfering with the others memory space. This tool is also used to ensure that only the code and data needed to run the game are being loaded into RAM at once. Page cache is used to store frequently accessed code and data, improving performance and reducing disk I/O. Using Memory Mapping developers can map files directly into a processes address space which improves performance. Control Groups allows administrators to limit resources used by certain processes, which can prevent overutilization for any particular process.
5. **Distributed Systems and Networks**: Implementing a client-server architecture where one or more servers will run the games logic, player interactions, and data storage. Clients from different platforms will access these same servers. Use HTTP for standardized communication protocol. Define clear/consistent APIs between clients and servers. Utilize cross-platform libraries to support multiple operating systems and devices. Load balance the servers so that traffic can be managed quickly and efficiently. Use HTTPS to protect sensitive information as data is transferred. Use authentication to ensure protected user access. Use horizontally scaling and cache database design for scalability. Implement monitoring and analytics for data regarding player behavior, system function, and identification of issues.
6. **Security**: Implement token based authentication for verification of user identity. Use HTTPS protocol that ensures protected communication and use encryption platforms for other platforms. Implement sandboxing to protect the application from other system components. Ensure protected user input with SSL (secure socket layer). Store passwords in strong one way hash algorithms and salt, where one random input is added before they are stored to ensure security before storage, in addition. Use Content Delivery Network(CND) Security available through the CDN.