

Draw It or Lose It Web Application

# **CS 230 Project Software Design Template**

Version 1.4

## 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 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

**Recommendations 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0  1.2  1.3  1.4 | 05/24/25  6/8/25  6/22/25  6/26/25 | AARON VOGEL  AARON VOGEL  AARON VOGEL  AARON VOGEL | Draft of Software Design Doc  Evaluate various platforms  Analyze system architectures  FINAL |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room partnered with Creative Technology Solutions and wants to expand from their mobile game(Draw It or Lose It), into the web based market. So this would mean use of the game on multiple platforms. To do this I would use Java with design patterns like Singleton and Iterator. That would help with scalability and maintenance.

## Requirements

Should support one or more teams.  
Multiple players per team.

Unique names  
Only one instance at a time  
Web based game working across other platforms also

## [Design Constraints](#_2et92p0)

Being web-based would require session handling and platform indepences. Distributed system must handle more than one client connection while only keeping one active game instance. The unique names would need a validation of names to stop any conflicts and to keep consistency across all devices and browsers.

## [System Architecture View](#_ilbxbyevv6b6)

To allow Draw It or Lose It to expand and run across multiple platforms, I recommend using **Linux** for the server platform. Linux is reliable, affordable, and works very well for web hosting. It is also widely used in the tech industry, making it easy to find support and resources. Because Linux is open-source, there are no expensive licensing fees, and it can handle large numbers of users without slowing down. This will help the game grow over time without running into performance problems or high costs.

The system architecture will use a web-based design that separates the frontend and backend. The backend will be built using Java and run on the Linux server. This part of the system will handle the game logic, store data, and manage communication between players. The frontend will be built with React, which allows the game to work on many devices, including Windows computers, Macs, smartphones, and tablets. This design allows for smooth operation no matter which device the player is using. The backend and frontend will communicate using REST APIs over secure internet connections, while WebSockets will handle the real-time parts of gameplay so players can see updates right away as they happen.

For storage, I recommend using MySQL as the database. MySQL is a trusted database system that can store user information, teams, scores, and game progress in an organized way. It keeps data safe and consistent, even if many players are using the game at the same time. If needed, MySQL can also be set up to handle larger amounts of data as the game grows.

In terms of memory management, Java is a good choice because it handles memory automatically through something called garbage collection. This means that as players join and leave games, the system will automatically clean up memory that is no longer being used. To make sure only one game instance runs at a time, the Singleton design pattern will be used. This helps reduce the amount of memory needed and keeps the system running smoothly. If more speed is needed, tools like Redis can be used to temporarily store game information in memory for quick access.

Since the game needs to work across different platforms and devices, it will be set up as a distributed system. The game servers will handle player requests through RESTful APIs for normal game functions, while WebSockets will be used to keep game sessions updated in real-time. A load balancer will spread out the work among different servers, so no single server gets overwhelmed. If one server has a problem, others can take over to keep the game running without downtime. Cloud services like AWS or Google Cloud can also help by placing servers closer to players, which makes the game faster for everyone.

Security is very important for protecting players’ information. All communication between the players and the servers will be encrypted using HTTPS, which prevents data from being intercepted. Players will log in securely using token systems like OAuth2 or JWT, which are widely used for safe online logins. Any personal or sensitive data will be encrypted both when it is sent and when it is stored. The system will also have checks in place to stop things like SQL injection, cross-site scripting, and other common attacks. Regular security checks and updates will be done to keep the system safe. Players will only be allowed to access the parts of the game they are supposed to, using role-based permissions.

## [Domain Model](#_8h2ehzxfam4o)

Entity – Base class with name and id.

Game, Team, Player – Player inherit Entity. Team is list of Player, Game is list of teams.

GameService – Game logic, Maintains Singleton, IDs and collections.

Inheritance – Entity Class is base for code reuse  
Encapsulation – Private and public getters/setters  
Singleton Pattern – Only one GameService Instance  
Iterator Pattern – Check Unique names

**"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)

|  |  |  |  |
| --- | --- | --- | --- |
| Platform | Server Side Deployment | Client Side Compatibility | Development Tools |
| Linux | ✔ Linux is used for web hosting. It supports server-based deployment using tools such as Apache, Docker, Nginx, and Node.js. Scales well and has low resources. | ✔ Fully supports web browsers like Chrome and Firefox. No native mobile support, but the web app can run in the browser. | ✔ Open-source tools, Teams with Linux experience are needed, but no extra cost. |
| Mac (macOS) | ✔ Can run server software like Apache, Nginx, or MAMP. Not recommended for production hosting due to cost and server limitations. | ✔ Runs Chrome, Firefox, Safari. Fully supports browser web apps. Can test mobile browser emulation. | ❗ Xcode is required for iOS development. Limited to expensive Apple hardware. Best used for development and not server hosting. |
| Windows | ✔ Supports IIS, Apache, and other server software. More expensive (Windows Server licenses) and more resource-heavy than Linux. | ✔ Supports Edge, Chrome, and Firefox. Fully compatible with web apps. | ❗ Visual Studio. Windows-specific tools may require extra costs. Some tools can be more expensive than Linux. |
| Android | ❌ Not a server. Strictly a client platform. | ✔ Already supported. The game runs like a native app or via a browser. | ✔ Android Studio. Web compatibility means there is no need for complete native rebuilding when using responsive web technologies. |
| iOS | ❌ Cannot be used for hosting servers. | ✔ Needs Safari and Chrome compatibility. A responsive web app must be tested carefully due to stricter rules from Apple. | ❗ Requires macOS and Xcode. iOS app store has stricter submission guidelines, which may slow down the launch time. Developers would need Apple devices |

✔ - Favorable  ❗ **-** Caution(slow launch time or extra costs) ❌ - Cannot be used

**Summary**

The server-side recommendation is to use Linux for hosting the web application because it's cost-effective, scalable, and well-supported.  
The client-side Development is focused on building a responsive HTML5 app that runs on all browsers and mobile devices. This would eliminate the need to maintain separate native apps.  
The Development Tools: The open-source tools needed (Git, Node.js, VS Code) for general development. Use Android Studio for Android testing and Xcode for iOS testing. A cross-functional team may be needed to cover all platforms.

## 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**: Linux would be the best choice for costs, performance and scalability.
2. **Operating Systems Architectures**: Web-based App using REST API backend hosted by a Linux server and frontend designed using React for multi platform compatibility.
3. **Storage Management**: MySQL to structure users, teams and game states.
4. **Memory Management**: Java for managing memory. The Singleton design will make sure only one instance is active and that will reduce the memory needed.
5. **Distributed Systems and Networks**: : RESTful API and WebSockets will maintain the real-time gameplay. Load balancers and server monitoring will take care of traffic and outages.
6. **Security**: HTTPS for Communication with server to keep data safe. Using either OAuth2 or JWT to make sure people are able to log in and access the game. All data stored or sent will be encrypted. Lastly add checks to make sure users are not doing anything that could harm the system.