

**Draw It or Lose It**

# **CS 230 Project Software Design Template**

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 |
| --- | --- | --- | --- |
| 1.0 | 08/17/2025 | John Swindell | First version of the software design document for Draw it or Lose it. Backend for web expansion. |

## [Executive Summary](#_sbfa50wo7nsh)

The main problem is that the current application is available only on Android mobile devices, but expansion is necessary to reach a broader audience. This expansion must take place via web-based platforms in order to be available across numerous devices not currently covered. I have designed and developed the foundation for this expansion. The application, written in Java, uses object-oriented principles and design patterns, and forms the robust and scalable backend this expansion will be built on. This will be the backbone of the web development process moving forward.

## Requirements

**Business Requirements**

* Expand the Draw It or Lose It game from an Android-only app to a web-based platform to serve multiple devices.
* Increase the game's market presence and reach a wider audience.

**Technical Requirements**

* The application has to support games consisting of one or more teams, with multiple players per team.
* Games and team names must be unique.
* The system must ensure only one instance of the game service can exist in memory to manage all games centrally.

## [Design Constraints](#_2et92p0)

The main constraint is that the application must be a web-based, distributed environment, which means the system has to handle multiple games and users at the same time. This also implies that Games and Teams must all have unique names to prevent issues with conflicting game instances. This requires a robust checking system before any new games or teams are created. The central hub and point of control, GameService, has to manage all game instances, and one and only one can exist at any given time.

## [System Architecture View](#_ilbxbyevv6b6)

This section is not applicable to this project.

[**Domain Model**](#_8h2ehzxfam4o)

Each class serves a very specific purpose, as it is best practice in object-oriented principles. The GameService class is the singleton controller. Its purpose is to be the sole proprietor and orchestrator of all game instances at any given time, and is created using the singleton pattern to prevent multiple. The Game class is the container for teams, keeping them separated within their respective instances. The Team is a container for players, and Player is an individual user. Entity is intentionally made an Abstract class, as it should not necessarily be implemented itself, but rather each of the classes: Game, Player, and Team should all ‘extend’ the Entity class. This is an example of Inheritance, and gives them all common attributes that should be shared, such as id and name, which avoids the duplication of code unnecessarily. Additionally, there is the composition of object-oriented programming involved. This relationship is represented by the ‘has-a’ precursor. For instance, a Game *has a* list of Teams, and a Team *has a* list of Players. This is the most basic and fundamental structure for building out larger systems, ensuring that inheritance and encapsulation are implemented where possible to avoid redundancy and logically represent relationships. The design patterns used to implement each class are also intentional. While I have already touched on the singleton pattern, there is one more: the Iterator pattern. This pattern is used in the addTeam and addPlayer methods to ensure name uniqueness in both, as the web application requires this constraint to function properly without conflict.

**"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** | Very rarely used for game servers, largely used for client devices. Servers are expensive as server hardware from Apple is difficult to come by. A much smaller game server-side ecosystem. Would limit options unnecessarily. | Linux is the industry standard for game servers. They are highly flexible, high in stability, free, open source, software is plentiful, and security is strong. Linux is a solid choice. | Windows is a fine choice, but it is atypical for game servers to run. Software is less widely available, licensing fees are necessary. Unless you use Microsoft products primarily, there is no real benefit over Linux. | Objectively not suitable for our requirements. A mobile device is the client, not capable of hosting a web application. Capable of connecting to one, but certainly not a server. |
| **Client Side** | The main client-side concern for macOS is making sure they have full compatibility with the Apple Safari browser. Since Safari uses a different rendering engine (WebKit) than other major browsers, dedicated testing is required to guarantee the application's features and layout perform as expected for Mac users. | The client-side strategy for Linux will focus on supporting the platform's most prevalent open-source browsers. The main target will be Mozilla Firefox, which is the default for many popular Linux distributions. Compatibility with Google Chrome will also be a priority due to its widespread use among developers and users. | The client-side strategy for Windows will target the most widely used browsers on the platform. This includes dedicated testing for Microsoft Edge, as the default Windows browser, and Google Chrome, due to its dominance in the market for browsers anyway. The goal is to make sure all users have a seamless and fully functional experience for Windows users. | For mobile devices, the primary concern is making a responsive design. The web application's layout must automatically shrink to smaller screen sizes for both phones and tablets. Testing will focus on the dominant mobile browsers like Safari on iOS and Google Chrome on Android. We must make sure that all the features are functional and the user interface is touch-friendly for those on mobile. |
| **Development Tools** | Java is a platform independent language, as it runs on a JVM or Java Virtual Machine. Development can be done on any number of IDEs, such as Eclipse, IntelliJ IDEA, or any IDE that allows for the installation of a JDK. A version control system such as Git should be used, and Maven should be used for project dependency management. | This column shares the same response as the first column. However, I will add that documentation will need to change between platforms. MacOS has a terminal similar to Linux, and their documentation will need to be different for the developers across systems. Terminal commands designed for Linux and Mac users will be different than in Windows. | This column shares the same response as the first as well. I will add a similar point to documentation regarding PowerShell and command prompt. While similar, documentation and line commands do indeed differ between the two. I would suggest standardizing the development environment to use the terminal within the chosen IDE for this reason, or simply Git Bash. | Since this is a web application, the same backend development tools will be used regardless of platform. However, client-side development and resting will heavily involve using browser developer tools to simulate mobile devices and mobile screens. The same set of tools can also be used to simulate network conditions. The development for mobile should ensure the page has a responsive, adaptive design. |

## Recommendations

1. **Operating Platform**: Linux is the gold standard for game servers, and my personal recommendation is based on evidence above. I would highly advise a popular distribution such as Ubuntu Server. It’s low cost, has a large community, security reputation, and can support a wide range of free and open-source software specifically designed for game servers.
2. **Operating Systems Architectures**: Linux uses what is called a monolithic kernel, which if efficient, and is modular by design which allows for flexibility. Linux has native support for multi user and multi taking environments, and is a basic requirement for a game sever which needs to host multiple game instances and requests from many players at once.
3. **Storage Management**: I would recommend a relational database system. A noSQL system like MongoDB, while more flexible, is not necessary at this time. We are storing very simple data, and reliability and speed are more important given this use case. My recommendation would be MySQL or PostgreSQL. Both will work, and both connect very well with Java using JDBC.
4. **Memory Management**: In this game server context, memory management is handled by both the OS and the application itself. For instance, the Linux kernel is responsible for managing the server’s overall physical health and virtual memory, in that it will need to properly allocate memory to processes as needed. However, the application within the JVM will request contiguous blocks of memory from Linux and manage it specifically for the game. The most specific and important feature here is Java’s automatic garbage collection. This handles the deletion of objects created by Java on the heap, freeing memory and preventing leakage automatically.
5. **Distributed Systems and Networks**: The game uses a client-server model. The application runs on a remote server; the user connects to this running instance through the web. A player on any device can connect their client using the HTTP protocol. The primary dependency here is simply a stable internet connection and an API endpoint. With a REST API structure, accessible sever, and stable connection, anyone can access the game.
6. **Security**: The security approach will be a standard, highly secure, and multi-layered approach, involving network security, sever security, and application security. Network traffic must be encrypted using HTTPS. This means all communication between the player and the server will be kept secure. The server must be kept up to date with the latest security software, security patches, and a firewall should be configured to block any unwanted access. Lastly, any secrets such as API keys, passwords, or user data should never be stored or hard coded anywhere in plain text. They should be protected using hashing and salting. All input from users must be validated on the serve to prevent and kind of SQL query injection attacks.