

<Draw It or Lose It>

# **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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <09/21/25> | Howon Pottinger | <Brief description of changes in this revision> |

**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 Gaming Room wants to take Draw It or Lose It from an Android-only app to a web-based game that works across all platforms. The main goal is to have it where multiple teams and players can join, keep names unique, and make sure the game service runs as a single instance.

The plan is not difficult: Use the Singleton pattern to run one game service only and use the iterator pattern to check list for duplicates when adding games, teams or players. A base Entity class will hold shared fields like id and name for consistency. This design gives a foundation for a scalable, easy to maintain web game.

## Requirements

* The game can have one or more teams
* Each team can have multiple players
* Every game, team, and player gets a unique id
* Game, and team names must be unique
* Only one GameService can exist at a time
* Use iterations to check for duplicates and lookups

## [Design Constraints](#_2et92p0)

* This app must run a in a web based, distributed set-up, so the server keeps the master game list and ID’s and not the client.
* If two people try to create a game or team at the same time, the server must block duplicates
* Names must be checked for uniqueness before new games, players and teams are added
* The code should be portable across Windows, Mac, Linux, and then later support mobile and web clients
* The system must use secure connections to protect user information between the server and devices
* So, the system should be scalable and observable: log activity, handle growth, and recover from dropped connections

## [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 UML shows Entity as the base class with an id and name. Game, Team, and Player all extend Entity to share the setup. GameService manages the list of games and hands out IDs, using the Singleton pattern. The relationships are one-to-many: GameService →Game →Team → Player.

The OOP principles shown are inheritance (shared Entity), encapsulation (private list and IDs),

Abstraction (Entity and GameService roles), and a little polymorphism with methods like toString() and getName().

**"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** | macOS is great for development but is not commonly used for production servers. It has limited enterprise hosting options and vendor support compared to Linux or Windows. While excellent for developer workstations, macOS is not a standard production server platform and would not be targeted for hosting the API. | Linux is the most common OS for web servers and is known for stability, security, and scalability. It offers mature packaging, automation, and monitoring that make deployment reliable with low resource overhead. It integrates easily with CI/CD and has broad cloud support (e.g., Ubuntu LTS, Amazon Linux), making it the best overall choice for hosting. | Windows Server offers strong enterprise integration, especially with Active Directory and other Microsoft services. Java and **Dropwizard** run reliably on Windows, and admins familiar with the platform can manage it easily. However, licensing costs and a heavier resource footprint than Linux make it a viable but less efficient alternative for hosting. | Mobile devices are not used as servers due to limited processing power, storage, and battery life. They act as clients that connect to the API and consume services provided by the backend rather than hosting them. |
| **Client Side** | macOS has a smaller desktop market share than Windows but remains important for creative professionals and developers. It’s essential to test compatibility with Safari and other macOS system features to ensure consistent performance and a smooth user experience across platforms. | Linux has a smaller desktop market share but is widely favored by developers and technical users. Applications should be tested in browsers such as Firefox and Chrome on Linux systems to ensure they run smoothly and provide the same functionality and user experience as on other platforms. | Windows has the largest desktop user base and the widest variety of hardware configurations. Because of this diversity, client applications must be thoroughly tested for compatibility across different Windows versions and browsers to ensure consistent performance and a reliable user experience for the majority of users. | Mobile devices, including iOS and Android, represent the largest client base for the application. The interface must be responsive, optimized for touch controls, and perform efficiently on smaller screens to ensure a smooth and consistent experience for mobile users. |
| **Development Tools** | macOS supports Java and Dropwizard development using tools such as Eclipse and IntelliJ IDEA. Developers can also use Xcode when creating iOS builds, making macOS a strong option for teams targeting both web and mobile environments. Xcode is free, while IntelliJ IDEA Ultimate is paid (the Community Edition is free). | Linux provides an excellent open-source environment for Java development and automation. It supports Eclipse, IntelliJ IDEA, and command-line tools such as Maven, making it ideal for continuous integration, testing, and server-side development. Eclipse and Maven are free and open source, while IntelliJ IDEA offers both a free Community Edition and a paid Ultimate version. | Windows supports multiple development environments, including Visual Studio, Eclipse, and IntelliJ IDEA. It integrates well with enterprise systems and version control tools, making it a dependable platform for developers in larger organizations. Visual Studio Professional and Enterprise editions are paid, while Eclipse and IntelliJ IDEA Community Edition are free. | Mobile development relies on dedicated SDKs and emulators such as Android Studio and Xcode. Testing on both real devices and emulators ensures the application performs smoothly and remains responsive across different screen sizes and operating systems. Both Android Studio and Xcode are free to use, though publishing native apps later would require a one-time Google Play fee and an annual Apple Developer subscription. |

## 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**: I recommend using Linux as the operating platform for the server that will run *Draw It or Lose It*. Linux is free, stable, and secure, and it works smoothly with Java programs like ours.
2. **Operating Systems Architectures**: Linux Architecture is a layered system. At the base is the hardware, followed by the kernel, which controls how the system talks to the hardware. Above that are the system libraries and shell utilities, which let users and programs give commands to the system. At the very top are the applications, such as our Java-based game server.
3. **Storage Management**: The Linux server will use Logical Volume Management (LVM) for managing storage. LVM provides adaptable disk management by letting the system resize partitions while it’s running, merge multiple disks into a single storage pool, and take snapshots for backups. This makes it less difficult to expand or manage storage without downtime and helps The Gaming Room keep data safe as the game and user base grow.
4. **Memory Management**: Linux practices techniques like virtual memory, demand paging, and memory mapping to handle memory efficiently for the *Draw It or Lose It* server. These methods authorize the system to use physical memory and disk space effectively, so the game can run without errors, even as more players connect.
5. **Distributed Systems and Networks**: *Draw It or Lose It* uses a distributed setup where web, mobile, and desktop clients connect to one main server through the internet. To handle connection issues, clients try again safely, and a load balancer directs traffic away from bad servers. The API uses health checks and queues background tasks so the game keeps running accurately even if a server goes down.
6. **Security**: The system uses TLS/HTTPS to secure data between devices, token-based logins for authentication, and hashed passwords with strong algorithms like bcrypt or Argon2. Input validation prevents attacks, and role-based access protects admin features.