

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 03/23/25 | Joseph Wilfong | Initial Draft |
| [1.1](#_Evaluation) | 04/05/25 | Joseph Wilfong | Small update to evaluations section |
| 1.2 | 04/20/25 | Joseph Wilfong | Finalized the recommendations section |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is seeking to expand their existing Android-only game, Draw It or Lose It, into a web-based, multi-platform game. To meet this goal, a scalable, object-oriented software design is needed to ensure maintainability, code reuse, and consistency across various platforms. The solution includes the use of design patterns, specifically the Singleton and Iterator patterns, to manage game state and implement certain constraints like unique naming for games and teams. A UML class diagram is available to help guide development, and implementation will follow all best practices to build a secure and efficient application. This document aims to outline the software design and architectural components that are necessary to begin development.

## Requirements

* Business Requirements:
  + Expand Draw It or Lose It to web-based and multi-platform environments
  + Maintain uniqueness of game and team names
  + Allow one or more teams per game, and multiple players per team
* Tech Requirements:
  + Only one instance of a game should exist in memory (Singleton pattern)
  + Validate uniqueness of game and team names (Iterator pattern)
  + Design must support distributed web-based environments
  + Use object-oriented principles for scalability and maintainability

## [Design Constraints](#_2et92p0)

The main design constraints for this project come from the need to run the game in a web-based, distributed environment. This includes making sure that:

* The application must support only one instance of the game in memory using the Singleton design pattern, which prevents multiple instantiations of the game object that could lead to inconsistent states.
* Game, team, and player names must be unique, requiring the use of iterators to traverse existing collections for validation.
* The system needs to be designed with scalability and extensibility in mind while minimizing tight coupling between the classes.
* Web compatibility and cross-platform consistency must be maintained, which implies the use of portable technologies and frameworks such as Java (backend), Spring Boot (RESTful), React (frontend), and Docker (containerization). These constraints impact the decisions around class design, memory management, and state handling in a network environment.

## [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 class diagram for this application illustrates key object-oriented relationships and principles. At the top level, the **Entity** class serves as a base class with shared attributes like id and name, which shows inheritance. The **Game**, **Team**, and **Player** classes inherit from **Entity**, promoting code reuse and encapsulation of shared behavior. The **GameService** class manages instances of games, teams, and players, and applies the Singleton pattern to restrict instantiation to a single object in memory. Associations between classes are shown using multiplicities, accentuating composition and aggregation relationships. Finally, the Iterator pattern will be used to traverse collections when validating name uniqueness. This design also supports modularity and efficiency while meeting the client’s software requirements.

**"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** | MacOS offers decent tools for web development but is not commonly used for hosting servers. It is better suited for dev environments than production deployment. It also comes with licensing limitations and higher hardware costs, making it less practical on a large scale. | Linux is the most widely used OS for web servers due to its stability, performance, and open-source ecosystem. Which is very ideal for production environments. It also has no licensing fees and integrates well with tech like Docker and Kubernetes, which makes it scalable for thousands of users. | Windows server is viable but tends to require more system resources and licensing costs. It is better used for organizations standardized on MS tech. It can support enterprise-scale apps, but the added cost for licensing and its heavier system footprint can be a disadvantage for startups or cost sensitive deployments. | Mobile devices are not good for server-side hosting because of their hardware limitations and power/resource constraints. They are strictly used as client devices and are not capable of handling web server responsibilities. |
| **Client Side** | Developing client applications on macOS requires tools like Xcode and often limits flexibility unless targeting Apple devices. It may increase development time if porting to other platforms. However, macOS is still useful for testing web applications like Safari, which is essential when ensuring cross-browser compatibility. | Linux offers flexibility for web-based client development, but market share for end users seems to be pretty small, which would limit user testing coverage. Still, it supports all major browsers and works well for internal QA environments and automated testing pipelines. | Windows is a popular platform for client applications and web browsers, which can reduce testing and deployment challenges. Its higher market share makes it a priority for browser compatibility, especially with Chrome, Edge, and Firefox. | You must account for OS fragmentation, screen sizes, and touch screens; this increases the cost and makes tests more complex. Making sure responsiveness and UI behavior is consistent across Android and iOS devices also adds to the testing and development burden. |
| **Development Tools** | Common languages include Swift, Objective-C, and JS (JavaScript). IDEs include Xcode, IntelliJ, and VS Code. Xcode is only available on Mac, so devs targeting iOS will need access to Apple hardware. While most tools are free, this hardware dependency can increase development costs. | Java, Python, and JS. IDEs include Eclipse, IntelliJ, and VS Code. These tools are largely open-source and free, reducing overall cost. Linux-based development environments also support DevOps workflows efficiently. One full-stack team can often develop and deploy from Linux using industry standard tools. | C#, Java, and JS. IDEs include Visual Studio, Eclipse, and NetBeans. While Visual Studio Community is free, Enterprise versions carry licensing fees. Developers may also be more familiar with Windows-based environments, lowering training overhead. | Java/Kotlin and Swift. Tools include Android Studio, Xcode, Flutter, and React Native. Developing for both iOS and Android may require two separate teams unless cross-platform solutions are used to share code. These tools are mostly free but may require specialized skill sets. |

## 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 is recommended for deploying the server-side component due to its reliability, security, and scalability. For development, a combo of Linux for backend, Windows or Mac for front-end, and mobile platforms should be considered based on dev preferences and the application requirements. It’s also free to use and works with modern tools like Docker, which makes it easier to deploy the app and keep things consistent across servers.
2. **Operating Systems Architectures**: The app will adopt a layered architecture, with a service layer that handles business logic and a presentation layer for the web interface. Linux will handle the backend, using a microservice or REST-based architecture. This setup helps split up the app’s different parts, so they can be worked on or updated without breaking everything. It also makes things easier to test and scale later on.
3. **Storage Management**: A relational database like PostgreSQL or MySQL is recommended to manage persistent data such as users, games, teams, and player records. These databases are well supported in Linux environments and allow easy querying and transaction management. PostgreSQL is a good option because it’s solid, handles large amounts of data well, and supports features like backups, user permissions, and replication if you need that.
4. **Memory Management**: The Linux OS handles memory efficiently with virtual memory, swapping, and garbage collection with Java’s JVM. The Singleton pattern in the application also minimizes memory footprint by avoiding redundant object creation. If the app grows or needs to handle more users, memory settings can be tuned easily. Linux also gives devs good control over how memory is used, which helps avoid slowdowns or crashes.
5. **Distributed Systems and Networks**: For this you would use a RESTful API for communication between front-end (web or mobile) and the back-end services. Load balancing, containerization, and proper failover handling will make sure there is adequate reliability and scalability. Each piece of the app can run on its own in a container and talk to others through the network, which would help with scaling and makes things more stable if part of the system goes down. Tools like Docker and Kubernetes can be used to manage this setup.
6. **Security**: Secure communication will be enforced by using HTTPS. Input validation, authentication (OAuth2), and data encryption will protect user data. Linux’s robust security features, along with application-layer security measures, will help keep data safe across all platforms. User passwords will be hashed, sessions will be token-based, and sensitive information can be encrypted. Logs and monitoring can be used to catch suspicious behavior and help respond to issues quickly.