

# 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 | 11/17/24 | Umar Asif | Initial submission of the software design document with completed sections: Executive Summary, Design Constraints, Domain Model, Evaluation, and Recommendations. |

**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 is developing a game application Draw It or Lose It and is targeting its extension towards the web-based distributed environment. In the suggested software solution, this Singleton design pattern will make certain that throughout the system, it maintains the singleton instance of GameService and offers efficient game management. Object-oriented design should be implemented in the system's domain model in order for code scalability and reusability to meet the current requirements as well as achieve future expansions. The paper will outline the design constraints, the domain model, system architecture recommendations, and strategies for secure distributed systems and storage.

## Requirements

* Develop a web-based distributed game application.
* Provide a system that supports cross-platform communication.
* Implement a scalable design that allows the management of multiple games, teams, and players.
* Ensure security and data protection.
* Utilize object-oriented programming principles to achieve maintainable code.

## [Design Constraints](#_2et92p0)

* **Web-Based Distributed Environment**: The system must function efficiently in a distributed setup, requiring reliable communication across platforms and devices.
* **Scalability**: The application must support the addition of new games, teams, and players without performance degradation.
* **Platform Independence**: The solution must cater to multiple operating systems and devices.
* **Security Requirements**: Protecting user data and ensuring secure communication between systems is critical.

## [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 provided defines the relationships between GameService, Game, Team, and Player classes, with Entity as a base class for shared attributes and behaviors.

1. **Relationships**:

* GameService manages Game objects, which in turn manage Team and Player objects.
* Each class demonstrates aggregation and composition relationships.

1. **Object-Oriented Principles**:

* **Encapsulation**: Attributes are private, accessed through public getters/setters.
* **Inheritance**: Entity provides a shared structure, reducing code duplication.
* **Polymorphism**: Overridden methods like toString() enhance flexibility in object behavior.
* **Singleton Pattern**: Ensures only one instance of GameService is active, simplifying resource management.

**"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 offers a Unix-based server environment, robust and secure. Tools like Apache and Nginx are available.  Excellent for development and testing; seamless integration with Apple technologies.  Limited use as a large-scale web server, higher hardware costs, less support in large-scale hosting environments. | Linux is the most popular server OS globally due to its flexibility, security, and community support.  Cost-effective, supports a wide range of server software, scalable for thousands of users.  Requires technical expertise to manage and configure. | Windows Server offers IIS (Internet Information Services) for hosting web applications.  Easy-to-use GUI, integration with Microsoft products, strong enterprise support.  Licensing costs are high, less stable for high-traffic environments compared to Linux. | Mobile devices access the server via browsers or apps. Direct hosting on devices is impractical.  Supports responsive design for browsers and devices.  Relies on the server's capability; mobile devices can't act as servers. |
| **Client Side** | Responsive web design ensures compatibility with Safari and other browsers.  Developers may need macOS systems for testing and debugging; hardware and expertise costs are high. | Support for Firefox and Chrome ensures compatibility; open-source tools assist in cross-platform testing.  Low cost but fewer users compared to Windows/Mac; expertise in Linux testing is necessary. | Broad compatibility with Edge, Chrome, and Firefox; tools like Visual Studio are useful.  Wide user base necessitates thorough testing; relatively lower hardware cost. | Ensure responsive design; extensive testing on iOS and Android for screen sizes and browsers.  Significant time and resources required for testing across many devices. |
| **Development Tools** | Xcode, Atom, Sublime Text, browser debugging tools.  Xcode is free; macOS hardware is expensive.  Teams may need dedicated Mac systems for testing and debugging. | Eclipse, VS Code, Node.js, Flask, open-source frameworks.  Minimal to none; most tools are free and open-source.  Ideal for budget-conscious teams but requires Linux expertise. | Visual Studio, .NET Framework, browser developer tools.  Visual Studio has free/paid versions; Windows licenses add costs.  Well-supported environment but requires investment in licenses. | Android Studio, Xcode (for iOS), React Native.  Free tools available; costs for testing on multiple devices.  Requires expertise in both iOS and Android development. |

**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**: Recommend Linux due to its scalability, cost-effectiveness, robust support for distributed systems, and broad compatibility with various server configurations.
2. **Operating Systems Architectures**: Recommend Linux-based microservices architecture. This modular approach allows different components of the game, such as user management, game logic, and image rendering, to function independently and communicate through RESTful APIs.
3. **Storage Management**: MySQL or PostgreSQL is recommended for relational data (e.g., user data, game state). A cloud-based storage solution (e.g., AWS S3 or Google Cloud Storage) can handle the large library of high-resolution images efficiently.
4. **Memory Management**: MySQL or PostgreSQL is recommended for relational data (e.g., user data, game state). A cloud-based storage solution (e.g., AWS S3 or Google Cloud Storage) can handle the large library of high-resolution images efficiently.
5. **Distributed Systems and Networks**: Recommend using WebSockets for real-time communication between platforms and REST APIs for broader compatibility. Implement a cloud-based distributed environment with load balancers to manage high traffic and ensure minimal downtime during server outages.
6. **Security**: Ensure user data protection with end-to-end encryption using HTTPS and AES. Employ OAuth 2.0 for authentication and secure API calls. Regularly update and patch the Linux platform to address vulnerabilities and implement firewalls to safeguard against threats.