

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 | 2/18/2025 | Blake Springman | Description of the code for the Draw It or Lose It game |

**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 seeks to develop a web-based version of their game, "Draw It or Lose It," to extend its reach beyond the existing Android app. This web-based game will enable cross-platform compatibility, engaging a broader audience and increasing revenue. The game involves teams competing to guess what is being drawn from a library of stock images. Each game consists of four rounds, with images fully rendered at the 30-second mark and a 15-second window for other teams to guess if the initial team fails. This document outlines the proposed software design, addressing key requirements and constraints to ensure seamless development and deployment.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple layers assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

* The game must function seamlessly on multiple platforms, including web browsers on both mobile devices (Apple and Android).
* The game must support multiple teams and players, with unique team names.
* Only one instance of the game exists in memory at any given time.
* The application must handle multiple concurrent users without performance degradation.
* Protects user data and ensure compliance with relevant data privacy regulations.
* Develop a robust backend infrastructure to support the game logic and data management.

## [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)

This UML diagram represents serves as a representation of the software to manage The Gaming Rooms web-based application. It is made of several classes, their relation to each other, and also demonstrating OOP principles.

The **Game**, **Team**, and **Player** classes inherit from the **Entity** class, which is indicated by the arrows pointing from these classes to **Entity.** **GameService** has associations with **Game**, and **Team** through the **games** attributes and methods like **getGames(), getGame(), addGame(), getTeams(), getTeam(),** and **addTeam().**  **Game** has an association with **Team** through the **teams** attributes and the **addTeam()** method. **Team** has an association with **Player** through the **players** attribute and the **addPlayer()** method.

Each class encapsulates its attributes and methods, providing public methods to interact with the private attributes. The **Entity** class serves as a base class for **Game, Team, and Player**, promoting a hierarchical relationship. The **GameService** class uses the singleton pattern as indicated by the **getInstance()** method and the private **service** attribute, ensuring one instance of **GameService** exists. The instance between all of these classes associates which allows for interactions between each other.

The efficiency is backed up by the encapsulation, inheritance between classes, the singleton pattern, and the association with these classes.

**"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** | **Characteristics:** Known for its stability and user-friendly interface.  **Advantages:** Stability, robust security  **Weaknesses:** Costly, limited hardware options | **Characteristics:** Robust, secure, flexible, highly versatile.  **Advantages:** Free to use, stability and very reliable, provides robust security features  **Weaknesses**: Steep learning curve (requires more training), compatibility issues | **Characteristics:** Extensive software compatibility and user-friendly, flexibility in hardware choices  **Advantages:** Wide compatibility, powerful developmental tools (Visual Studio), robust support, built-in security features  **Weaknesses:** Licensing fees, higher resource consumption, security vulnerabilities, intrusive updates and reboots. | **Characteristics:** highly portable, contains two main OS(Apple and Android).  **Advantages:** allows users to access web-based features on the go, touch interaction, widely used across the globe, gives real-time updates and notifications.  **Weaknesses:** Less processing power and memory, reliance on battery power, network dependency, and security concerns |
| **Client Side** | Costs: hardware, software licenses, development tools and infrastructure)  Time: Learning curve, development and testing, and app store approval  Expertise: Development skills, design and UI, and testing | Cost: cost-effective hardware choices, no licensing fees for Linux OS, free open-source development tools, and potential costs for cloud services  Time: Training and upskilling for proficiency, development and testing across different distributions, and careful deployment and maintenance planning. | Costs: Potentially cost-effective hardware choices, OS licenses, development tool licenses (Visual Studio).  Time: Specific training and upskilling for specific tools and requirements, careful deployment and maintenance planning.  Expertise: Proficient in windows programming languages, best practices in security, system administration skills for Windows servers. | Costs: Device costs, developer fees, potential third-party library costs, project management and CI/CD tool costs, and cloud service expenses.  Time: Designing for different screen sizes, regular updates and maintenance, app store updates and approval.  Expertise: Proficiency in programming languages such as swift, java, security best practices, backend integration, and comprehensive testing skills. |
| **Development Tools** | Languages: Swift, Objective-C, Python, JavaScript, HTML, and CSS  IDE’s: Visual Studio Code, Xcode, Node.js, Atom, Git, GitHub | Programming Languages: C++, Python, JavaScript, Java, PHP, Ruby, and Rust  IDE’s: VS Code, Eclipse, PyCharm, Atom, Docker, GitHub/Gitlab | Programming Languages: C#, JavaScript, Python, C++, Java, HTML/CSS  IDE’s: VS Code, Eclipse, PyCharm, Node.js, GitHub/Gitlab, Docker | Programming Languages: Swift, Objective-C, Kotlin, Java, Dart, JavaScript, C#  IDE’s: Xcode, Android Studio, VS Code, React Native, Flutter, IntelliJ IDEA, Firebase |

## 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 an appropriate OS to use for its robust, flexible, and cost-effective platform for expanding Draw It or Lose It. The stability, scalability, and extensive tool set would ensure a seamless experience of the game and meet business and technical requirements as well.
2. **Operating Systems Architectures**: Linux is a highly versatile operating system that supports a wide variety of hardware architecture, making it an ideal choice for hosting web-based software applications.
3. **Storage Management**: Logical Volume Manager or LVM would be a good use for storage because it provides a flexible and powerful way to manage storage devices and partitions.
4. **Memory Management**: Linux uses a variety of memory management techniques to efficiently manage system resources and ensure optimal performance of software applications. By using memory management techniques, Linux ensures efficient use of system resources, enhances application performance, and maintains system stability. In this case, Draw It or Lose It, these techniques can help the game run smoothly and provide reliable user experience.
5. **Distributed Systems and Networks**: To expand Draw It or Lose It across multiple platforms, Linux is recommended for its flexibility, cost-effectiveness, and robustness. Linux’s support for various architectures (x86, ARM) ensures compatibility for diverse hardware. Utilizing Logical Volume Manager (LVM) for store management and employing advanced memory management techniques enhances performance and stability. A microservices architecture, coupled with APIs, message queues, databases, facilitates seamless communication between web, mobile, and desktop clients. Ensuring reliable network connectivity, security, fault tolerance, and monitoring are critical for maintaining a responsive and secure system. This approach enables efficient development, deployment, and scalability, providing a seamless user experience across different environments.
6. **Security**: To protect user information while expanding the game on multiple platforms, leveraging Linux is highly effective due to its robust security features, such as strong user permissions, regular updates, and built in encryptions. Key strategies include using TLS for encrypting data in transit and tools like LUKS for data at rest, implementing multi-factor authentication and role-based access control, designing secure APIs and anonymizing user data. Employing regular security audits, penetration testing, and comprehensive logging helps identify and address vulnerabilities. Additionally, keeping software up to date, following secure coding practices, and informing users about security best practices ensures a secure environment across various platforms, safeguarding user information effectively.