

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 | 10/02/23 | Johnny Lewis | Listing and providing a brief and detailed summary of design requirements and limitations. |

**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)

Our client the gaming room wants to develop a new web-based game based on their current one called, Draw it or Lose It. Our client wants their new game to be accessible to multiple platforms. This game's objective is to have numerous teams made up of several players go through four rounds at a minute each. One team makes estimates up until the timer goes off when a photo is selected from a library of pictures. If not, until the allotted 15 seconds have passed, each member of the opposing side may respond.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players 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. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

The requirements provided by the client are to try and ensure that the application is developed to the standards they deem to be correct. Many of these requirements, such as the number of game instances that can exist, are to ensure that the game runs efficiently and functions appropriately as intended. If the game is to run on multiple platforms as intended, we need to ensure that a user or team can’t run the application on multiple devices at the same time. For example, if a player or team were to run the application on windows but then run it as well on their mobile device, many problems or errors might occur affecting live accurate account of player and team data, and memory management of the webpage.

## [Design Constraints](#_2et92p0)

* **Performance**: The application should be very responsive to user input and allow for minimal bugs and errors within the application.
* **Accessibility:** The application should provide features that allow users with disabilities to access the game,
* **Security:** The application should provide secure and safe transmission of user data. It should also have implementation of security features that prevent web-based attacks such as denial-of-service and SQL injection for back-end.
* **Scalability:** The application should support large amounts of data being transferred from user to host. It should also support the ability to maintain many users playing at a given moment.
* **Compatibility:** It should be able to be accessed from a multitude of web browsers such as Microsoft Edge, Opera, and Google Chrome.

## [Domain Model](#_8h2ehzxfam4o)

**The Entity class in the domain model has traits and behaviors that are shared by all other classes. It illustrates the inheritance concept of object-oriented programming by enabling other classes to take on its attributes and methods.**

**The Player class describes a player inside the game application. It creates a string that is used as a name for the player. This is referenced by the Team class and adds the string name of the player to a list of players known as a team.**

**The Team class has a string for the name of the team that the players make up. It has a function that adds a player by referencing the player’s name and adds it to a list of players. The list is used to describe what a team is.**

**The Game class references the Team class. In the game multiple teams are competing with each other, so the Game class references the Team class which in turn references the player class. All of these classes together use aggregation in conjunction with each other to form complex concept or class which is the GameService class.** Each class can be described as multiple parts of a system that work together to form a finished product.

**"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** | **Characteristics:** Unix-based, POSIX compliant, reliable, secure.  **Advantages:** Excellent for web development, supports popular web technologies.  **Weaknesses:** Limited to Apple ecosystem, expensive hardware. | **Characteristic**s: Open-source, flexible, customizable, high stability.  **Advantages:**  Ideal for web servers, great community support.  **Weaknesses:**  May require more technical expertise, less user-friendly. | **Characteristics**: Widely used in the enterprise, extensive tool support.  **Advantages:** Supports many web technologies, compatible with many applications.  **Weaknesses**: Limited native development tools, licensing costs. | **Characteristics**: Diverse platforms (iOS, Android), ecosystem of app stores.  **Advantages**:  Large user base, widespread availability.  **Weaknesses**: Fragmentation across devices and OS versions, security challenges. |
| **Client Side** | **Cost**:  Developing software for Mac clients may involve licensing fees for Apple developer programs and macOS-specific development tools. Acquiring Apple hardware for testing and development can be costly.  **Time**: Mac applications require adherence to Apple's design guidelines and approval process, which can add time to the development lifecycle.  **Expertise**: Development expertise in macOS-specific programming languages like Swift and Objective-C is essential. Familiarity with Apple's ecosystem and platform-specific APIs is necessary for optimal performance and user experience. | **Cost**:  Linux is an open-source platform, which reduces licensing costs. However, ensuring compatibility across various Linux distributions may increase development and testing expenses.  **Time**:  Linux development may take longer due to fragmentation among distributions and the need for thorough compatibility testing.  **Expertise**: Expertise in Linux development environments, command-line tools, and package management is necessary. Knowledge of different libraries and dependencies on various distributions is vital for seamless software support. | **Cost**:  Windows development may require Microsoft developer program subscriptions and licenses for certain tools. However, Windows PCs are widely accessible, reducing hardware costs for testing.  **Time:**  Developing for Windows can be relatively fast due to its comprehensive development ecosystem and extensive documentation.  **Expertise**: Proficiency in Windows-specific programming languages like C# and knowledge of the .NET framework are essential. Familiarity with Windows design guidelines and APIs ensures a consistent user experience. | **Cost**:  Developing for mobile devices involves membership costs for Apple's App Store and Google Play Store, along with potential fees for mobile development tools.  **Time**:  Developing for multiple mobile platforms can extend development time, as each platform requires separate codebases and UI considerations.  **Expertise**:  Expertise in platform-specific languages is vital. Knowledge of mobile-specific design patterns and guidelines ensures a polished app experience on each platform. |
| **Development Tools** | **Programming Languages:**  Swift and Objective-C are the primary programming languages used for macOS and iOS development. Swift is now the preferred language for macOS and iOS app development due to its modern syntax and performance.  **IDEs and Tools:**  For macOS and iOS development, Xcode is the official Integrated Development Environment (IDE) provided by Apple. Xcode includes tools like Interface Builder for designing user interfaces, Instruments for performance profiling, and other essential utilities for building, testing, and debugging macOS and iOS applications. | **Programming Languages:**  A wide variety of programming languages can be used for Linux development, such as C, C++, Python, Java, Go, and more.  **IDEs and Tools:** Popular IDEs for Linux development include Visual Studio Code, Eclipse, and JetBrains' IntelliJ IDEA. However, many Linux developers prefer to use lightweight text editors like Vim or Emacs combined with command-line tools for compilation and debugging. | **Programming Languages:**  The primary languages for Windows development are C# and C++ with the .NET framework. C++ is commonly used for system-level applications, while C# is widely used for Windows desktop and UWP (Universal Windows Platform) applications.  **IDEs and Tools:** Visual Studio is the official IDE for Windows development, offering comprehensive support for C#, C++, and other languages. Visual Studio provides powerful debugging tools, design editors, and integration with Microsoft Azure for cloud-based applications. | **Programming Languages:**  For iOS development, Swift and Objective-C are used, while Android development primarily involves Java and Kotlin.  **IDEs and Tool**s:  For iOS development, Xcode is the go-to IDE, as it offers seamless integration with Apple's development ecosystem and debugging tools.  For Android development, Android Studio is the official IDE, supporting Java and Kotlin with advanced features like layout editors, device emulators, and APK (Android Package) management. |

## 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 an excellent operating system choice. It boasts remarkable stability, security, and efficiency. Programs run seamlessly on Linux and offer extensive customization in their development. Being open-source, Linux gives developers the freedom to alter, access, and share it, leading to enhanced maintainability and quicker updates.
2. **Operating Systems Architectures**: The Linux operating system is designed to work with a wide variety of computer hardware, from personal computers and smartphones to large, specialized servers. Its adaptability allows it to operate on both standard and unique hardware configurations. This flexibility is a key reason for Linux's broad acceptance as a reliable and efficient operating system.
3. **Storage Management**: Nearly every Linux distribution comes equipped with LVM, a robust and flexible storage management solution. LVM offers features such as resizing volumes on-the-fly, creating snapshots for backups, and seamlessly transferring data across multiple physical devices, providing administrators with enhanced disk space management capabilities.
4. **Memory Management**: Linux's memory management strategies ensure that web applications operate smoothly and can scale to accommodate many users. These techniques, geared towards efficient and secure RAM utilization, include virtual memory and buffer caching to boost speed. Furthermore, access control mechanisms are employed to maintain system stability and security, even under heavy memory usage.
5. **Distributed Systems and Networks**: APIs enable various software components to interact over a network through a defined set of rules and protocols. Network protocols such as HTTP/HTTPS or TCP/IP provide standardized pathways for reliable data exchange between devices. To ensure seamless operation and avoid communication disruptions, it's crucial to manage the interdependencies of these software components. This involves overseeing how they relate to and engage with each other, utilizing tools like service discovery systems and configuration management.
6. **Security**: In a Linux system, user information is safeguarded using data encryption tools like LUKS for data at rest and SSL/TLS protocols for data in transit. Additionally, robust user authentication and authorization are paramount, employing techniques such as multi-factor authentication (MFA) for verifying user identities and role-based access control (RBAC) to delineate user permissions. To bolster network security, firewalls like iptables are implemented, establishing a robust defense between trusted and untrusted networks.