

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 | 11/12/23 | Tyler Ellis | Initial Software Development Plan |

**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 has tasked Creative Technology Solutions with developing a web-based application variant similar to the pre-established android game "Draw It or Lose It." The current objective is to translate the game from an Android only based application to a multi-platform, web-based application. Our task includes developing a scalable, convenient, and user-friendly web application that retains the underlying gameplay experience while giving access across various platforms.

Our proposed solution's basic functions include:

* Development of a responsive web application that is compatible with numerous devices and browsers.
* Development of a solid back system to manage game logic, user interactions, and data storage.
* Generating unique identifiers for games, teams, and players to better manage game instances.
* Prioritization of user experience with an intuitive interface and flawless gameplay.

This project will not only expand "Draw It or Lose It's" reach and engagement, but it will also set the foundation for future expansions and developments.

## Requirements

* Multi-Platform Support: The game must be accessible on various web browsers and devices.
* Unique Identifiers: Each game, team, and player must have unique identifiers to prevent conflicts.
* User Interface: The interface should be intuitive and user-friendly.
* Performance: The game should load quickly and run smoothly across all supported platforms.
* Scalability: The system should be able to handle more users and game instances.
* Data Management: Efficient handling and storage of game data, user profiles, and game history.

## [Design Constraints](#_2et92p0)

* **Cross Platform**: Ensuring consistent performance and appearance across different browsers and devices.
* **Unique Identifiers:** Developing a robust system to generate and manage unique identifiers for games, teams, and players.
* **Load Balancing**: Balancing server loads, especially during peak usage times.
* **Responsive Design**: Creating a UI that adapts to different screen sizes and resolutions.
* **Data Security and Privacy**: Ensuring the security of user data and compliance with data protection regulations.
* **Scalability**: Designing the backend to efficiently scale with the growing number of users.

## [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 domain model for "The Gaming Room" web application is outlined in the provided UML class diagram, demonstrating a structured approach to describing the game "Draw It or Lose It." The Entity class is essential to the model. It acts as a base class encapsulating common attributes like id and name, enabling code recycling through inheritance by derived classes such as Game, Team, and Player. These derived classes establish a hierarchical and relational structure so that a Game can have numerous Teams, each of which can have several Players, ensuring that the software satisfies the requirement for multiple teams and players within a game. Additionally, the diagram indicates the use of the Singleton pattern, which is enforced by the SingletonTester class. This is to keep a single instance of a game in memory which addresses the client's requirement for unique game instances whilst reflecting object-oriented principles of encapsulation and abstraction for system efficiency and maintainability.

**"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 must 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** | Mac servers are known for their stability and integration with Apple's ecosystem. They provide support for web-based applications, particularly those designed with Apple's own development tools. However, Mac servers can be more expensive, and the available software for server management might not be as diverse as on Linux or Windows. | Linux is widely used for its stability, security, and flexibility, making it a popular choice for web servers. It's also open source, which can significantly reduce costs. However, Linux requires more technical expertise to manage, and certain proprietary software might not be natively supported. | Windows servers offer easy integration with other Microsoft products and services, which can be a significant advantage for certain web applications. They are user friendly but can be costlier due to licensing fees and are typically considered less secure and stable than Linux servers. | Mobile devices are not traditionally used for hosting web-based software applications due to their limited processing power, storage, and uptime reliability compared to dedicated servers. They could serve as temporary or ad-hoc solutions but are not recommended for professional hosting solutions. |
| **Client Side** | Developing Mac clients requires consideration for the Apple ecosystem. Tools and applications must be compatible with macOS, which can incur additional costs for development licenses. Development might be faster due to integrated tools like Xcode, but this necessitates macOS expertise. | Supporting Linux clients can be cost-effective since many tools and applications are open source. However, developers must consider the fragmentation of Linux distributions which can increase development and testing time. Expertise in various Linux environments is also necessary. | Windows has a vast user base, which necessitates support for multiple versions, increasing testing and development complexity. Development tools and licenses can be costly, but a wide array of development tools and a large developer community can reduce development time. | Developing for mobile clients entails dealing with a variety of screen sizes and hardware specifications. The development process can be costly due to the need for multiple device testing and possible licensing of mobile development platforms, but the ubiquity of mobile devices makes them an important platform to support. |
| **Development Tools** | For Mac, Swift and Objective-C are common programming languages, with Xcode being the primary IDE. Other tools include Homebrew for package management and Git for version control. These tools are robust but can limit development to Mac environments. | Linux development often utilizes languages like Python, Java, and C/C++, with IDEs such as Eclipse and IntelliJ IDEA. Other tools include Git, Docker, and various package managers like APT and YUM, allowing for a versatile development environment. | On Windows, languages like C#, Visual Basic, and F# are common, with Visual Studio being the preferred IDE. Other tools include Microsoft SQL Server for databases and PowerShell for scripting, providing a comprehensive development ecosystem. | Mobile software development typically involves Java or Kotlin for Android and Swift or Objective-C for iOS. IDEs include Android Studio and Xcode, respectively. Cross-platform tools like React Native or Flutter are also widely used to streamline development across devices. |

## Recommendations

Analyze the characteristics of techniques specific to various systems architectures and recommend to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: A multi-platform solution is suggested for "The Gaming Room" in order to expand "Draw It or Lose It" across various technological platforms. The game can be accessible from an assortment of devices, regardless of the hosting operating system, thanks to the use of cloud-based infrastructure and web services. A combination of Linux-based backend servers and HTML5 for the client side is recommended. Linux was chosen for its reliability, scalability, and relatively low cost, while HTML5 enables the game to run on any modern web browser across desktop and mobile devices.
2. **Operating Systems Architectures**: Linux features a modular architecture, with a monolithic kernel controlling vital system functions and processes and an assortment of user space applications providing additional services. It is versatile for server-side operations since it supports a wide range of hardware and networking protocols. HTML5 is platform-agnostic on the client side and communicates with the system via the browser, which works as an intermediary, enabling device compatibility and responsive design.
3. **Storage Management**: A relational database management system, like PostgreSQL or MySQL, is excellent for maintaining game and user data due to its resiliency and consistency. It provides the functionality necessary to enforce unique name standards and manage an enormous number of operations, which is critical for a game with many concurrent users.
4. **Memory Management**: The Linux operating system employs advanced memory management techniques such as virtual memory with paging and swapping to efficiently handle the game's memory requirements. Garbage collection in backend services should be implemented if using languages like Java or Python to prevent memory leaks and ensure that only one instance of the game is in memory.
5. **Distributed Systems and Networks**: A microservices architecture, where independent gaming applications are deployed independently but interact together via APIs over a network, can be utilized to optimize communication across platforms. Load balancers, redundant network connections, and CDN services all have the potential minimize connectivity issues and disruptions, resulting in a more seamless user experience.
6. **Security**: Implementing HTTPS for secure communication, leveraging OAuth for authentication, and sticking to best practices such as frequent security audits and enforcing strong password restrictions can all help to protect user and company information. To safeguard against threats on the server, security measures consisting of firewalls, intrusion detection systems, and periodic updates are essential. Data encryption at rest and in motion assures that even if unauthorized access has been gained, the data remains unreadable.

Multi-Team Capability: The game needs to support one or more teams.

Team Size: Each team should/can have multiple players.

Name Standards: Both game and team names must be unique. Users must check if a name is already in use or available.

Game Instance Memory Management: Only one instance of the game needs to exist in memory at any given time. This will require unique identifiers for each game, team, or player instance.