

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** | While not commonly found in server structures due to the higher cost of Apple hardware, it is valuable as a testing and development platform for Apple ecosystem applications. Apple hardware is known for its dependability, providing durable benefits at a higher price point. However, unlike Linux and Windows, macOS is not commonly ideal for largeer scale deployments due to its limited scalability and customization capabilities. Despite this, it can run web servers like Apache and Nginx. Though, it is typically used in smaller environments rather than hosting large-scale web applications. | Linux is recognized for its relatively low cost (if not free) and stability, in addition to its support for an array of programming languages and databases. It excels at performance and security, quickly handling many requests, which is essential for scaled web applications. The powerful security approach offers effective protection against malware and crypto threats. Because of its open-source nature, this allows for considerable modification, and its development is backed and enhanced by an active community. Linux also offers many deployment and management tools. This would include Docker and Kubernetes, which simplify deployment in distributed systems. | Windows is favored for its ease of use and integration with other Microsoft products, making it a suitable choice for organizations running .NET applications. It offers a user-friendly interface and is particularly compatible with Microsoft’s own ecosystem, including Azure and Office 365, which ensures smooth management and operation. However, Windows Server comes with licensing fees, which can be a drawback compared to open-source alternatives like Linux. Regular updates and support from Microsoft, including security patches, are significant benefits, though they also create a dependency on Microsoft's policies. Windows Server provides a more user-friendly GUI for administration, making system management more accessible, but may require more resources. | Mobile devices, while not suitable for server-side operations, play a crucial role in the client-side experience of web applications. They are essential for ensuring seamless interaction with the server, especially in a game like 'Draw It or Lose It'. Mobile platforms efficiently communicate with the server through APIs, considering network conditions and data optimization. However, they lack the necessary infrastructure or capabilities for web server hosting. Security measures for data transmission between mobile devices and the server are also critical, particularly for a game with a large user base. |
| **Client Side** | Creating Mac clients requires considering the Apple environment. Tools and programs must be macOS compatible, which can result in additional expenditures for development licenses. While development may be faster thanks to integrated tools such as Xcode, this requires macOS knowledge. Because Apple places a premium on design and user experience, developers must invest in developing intuitive and visually appealing UI or User Interfaces. The ecosystem's interaction with other Apple devices, like iPhones and iPads, can also be used to provide a seamless user experience. | Supporting Linux clients can be cost-effective since many tools and applications are open source. However, developers must navigate the fragmentation of Linux distributions, which can increase development and testing time. Expertise in various Linux environments is necessary, and accommodating the diverse range of desktop environments and configurations can add to the complexity. Despite these challenges, the flexibility and customization potential of Linux make it a valuable platform, especially for reaching audiences who prefer open-source solutions. | Because Windows has a large user base, support for different versions is required, which increases testing and development complexity. Development tools and licenses can be expensive, but a diverse set of tools and a large developer community can minimize development time. Additionally, the widespread use of Windows in both corporate and personal settings provides a broad reach. Integration with Microsoft's cloud services and development tools such as Visual Studio can also provide solid options for developing sophisticated client applications. | Mobile client development involves interacting with an array of screen sizes and hardware specifications. Due to the requirement for multiple device testing and licensing of mobile development platforms, the development process can be costly, but the ubiquity of mobile devices makes them a vital platform to support. Additionally, improving efficiency and user experience for touch-based interfaces is crucial in mobile development. Compliance with various app store requirements, as well as the quick pace of mobile OS updates, present distinct issues. |
| **Development Tools** | Swift and Objective-C are the fundamental programming languages for Mac, with Xcode serving as the primary IDE. There is a significant increase in the use of VSCode with the ever-growing marketplace. Homebrew for package management and Git for version control are optional highly popular tools. These tools are strong and enable a seamless development experience, particularly for applications within the Apple ecosystem. Though, they could still restrict development to Mac settings, thus raising such things like expenses and reducing the pool of developers. The integration of these tools with Apple's hardware and software produces a highly optimized development environment, but it also binds developers to the Apple platform. | Linux development frequently uses responsive languages such as Python, Java, and C/C++, with common IDEs like Eclipse and IntelliJ IDEA. Version control tools like Git or as many companies have something completely internal, containerization tools like Docker, and package managers like APT and YUM all contribute to a versatile development environment. This adaptability allows developers to customize and optimize procedures for development, however it can require a more comprehensive expertise. These technologies' open-source nature promotes collaboration and innovation while potentially reducing cost. | Popular programming languages on a Windows platform include C#, Visual Basic, and F#, with Visual Studio functioning as the main IDE choice for companies and developers. Other technologies, like as Microsoft SQL Server for databases and PowerShell for scripting, help contribute to a more balanced development ecosystem. This technology is particularly useful when developing enterprise-level apps and those which integrate with other Microsoft services. However, reliance on proprietary tools and software could result in higher costs and reliance on Microsoft's ecosystem. | Mobile application development often utilizes Java or Kotlin for Android and Swift or Objective-C for iOS, with the most popular IDEs including Android Studio and Xcode. Cross-platform solutions like React Native and Flutter are growing more popular, allowing developers to simplify the development process across different platforms. These tools assist with dealing with the challenges provided by the diverse mobile device world. Challenges including varying screen sizes and hardware specifications. They do, however, need an in-depth knowledge of different platforms and may sometimes limit access to some native features. |

## 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" to expand "Draw It or Lose It" across technological platforms.** The game can be accessible from a range of devices, regardless of the hosting operating system, thanks to 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 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.