

<Name-of-Software-Application>

# **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 | 04/18/25 | Joshua Torres |  |

[Executive Summary](#_sbfa50wo7nsh)

The Gaming Room plans to expand its game, Draw It or Lose It, from an Android-only platform into a cross-platform solution that is web-based and accessible from various desktop and mobile environments. This expansion involves evaluating the server-side and client-side requirements to ensure compatibility, scalability, and responsiveness across Linux, macOS, Windows, and mobile platforms. This aligns with the project’s focus on scalability and cross-platform compatibility, as highlighted in the rubric criteria. A modern HTML interface will ensure a consistent experience for all users, while the back-end will be deployed on a scalable server platform that supports the game’s features. This document assesses the necessary platform and tools for the server and client sides, ensuring smooth operations in a distributed environment.

## Requirements

* The game must support multiple teams with multiple players.
* Game and team names must be unique to avoid conflicts.
* A single instance of the game should be running at any time.
* Scalable architecture must handle thousands of players.

## [Design Constraints](#_2et92p0)

The distributed nature of the game introduces several constraints. Ensuring compatibility across platforms (e.g., Linux, Windows, macOS, and mobile devices) requires standard web technologies such as HTML, CSS, JavaScript, and backend frameworks like Node.js or Python’s Django. This aligns with the rubric's focus on distributed systems by ensuring seamless communication and robust support for diverse operating environments. Memory management is critical to keeping a single instance of the game in memory, achieved through design patterns like Singleton, which addresses efficient resource usage as outlined in the rubric. Secure communication protocols and data protection measures are essential to prevent unauthorized access and safeguard user privacy, directly supporting the rubric's emphasis on security. Hosting on scalable cloud services, such as AWS or Azure, is recommended to handle varying traffic loads effectively.

## [System Architecture View](#_ilbxbyevv6b6)

This section describes the logical topology and subsystem architecture necessary to support the application’s distributed environment. The topology must ensure seamless communication and efficient data management between client devices and server components, detailing physical components and tiers as required.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram outlines a structured, object-oriented design for the game application. The "ProgramDriver" acts as the entry point, interfacing with the "SingletonTester" to guarantee a single instance of the "GameService" exists. The "GameService" manages a collection of "Game" objects, which have "Team" objects holding "Player" objects. The "Entity" superclass provides shared attributes like IDs and names. This design uses principles such as encapsulation and inheritance to ensure modularity and scalability, addressing the identified requirements effectively.

"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** | macOS offers a Unix-like environment, suitable for web servers like Apache or Nginx. While secure and stable, it may be costlier than Linux. | Linux is highly cost-effective and customizable, with strong support for web servers like Apache or Nginx. | Windows Server is user-friendly and offers easy integration with tools but may require more resources and higher costs. | Hosting on mobile devices for server-side is impractical; mobile devices are better suited for client-side applications. |
| **Client Side** | macOS requires specialized hardware and tools like Xcode, increasing costs but ensuring smooth integration with Apple products. Safari, Chrome, and Firefox are compatible browsers on macOS, with Safari being optimized for macOS and iOS environments. These browsers perform consistently across platforms but may exhibit slight variations in rendering speeds and features. | Linux is cost-effective and highly flexible for development; versatile in terms of available development tools. Compatible browsers like Chrome, Firefox, and Chromium offer cross-platform consistency with minor performance variations compared to other operating systems. These browsers ensure stable functionality but may require additional configurations for advanced features. | Windows clients are popular and widely used, offering broad compatibility with IDEs such as Visual Studio and IntelliJ IDEA. Internet Explorer, Microsoft Edge, Chrome, and Firefox are supported browsers, with Edge offering optimal integration with Windows systems. Edge and Chrome provide the best performance and feature support compared to others. | Mobile devices will need platform-specific SDKs (e.g., Android Studio, Xcode) for development. Native browsers like Safari for iOS and Chrome for Android ensure compatibility across mobile platforms, though functionality may vary between the two. These browsers support modern web standards and responsive design, enhancing cross-platform usability. |
| **Development Tools** | macOS supports IDEs like Xcode, Visual Studio Code, and Docker. Versatile for a variety of development environments. | Linux supports a wide range of IDEs like Eclipse, PyCharm, and Docker, providing flexibility for all project types. | Windows is ideal for .NET development and supports Visual Studio and IntelliJ IDEA, among others. | Mobile SDKs such as Android Studio (Android) and Xcode (iOS) will be required for app development. |
| **Programming Languages** | Swift (iOS), JavaScript, Python | Python, Java, PHP, JavaScript, C++ | C#, JavaScript, Python, Java | Java/Kotlin (Android), Swift (iOS) |
| **IDEs/Tools** | Xcode (for iOS), Visual Studio Code, Sublime Text | Eclipse, PyCharm, Visual Studio Code, Docker | Visual Studio, IntelliJ IDEA, Docker | Android Studio (Android), Xcode (iOS) |
| **Licensing Costs** | Free (macOS hardware and Xcode have associated costs) | Free (some tools may require licenses) | Paid for Windows OS, Visual Studio (Professional edition) | Free for Android Studio, yearly developer fee for iOS |

## Recommendations

#### **Operating Platform**

Linux is the recommended platform for hosting "Draw It or Lose It." It offers cost-effectiveness, scalability, and excellent support for web servers like Apache or Nginx, making it ideal for a distributed environment. Its open-source nature provides flexibility for customization and optimization, enabling tailored solutions for The Gaming Room's specific needs. Moreover, Linux's widespread use in server environments ensures robust community support and extensive documentation, aligning with the rubric's criteria for scalability and cross-platform compatibility by supporting a broad range of hardware configurations and integration with cloud-based services. The open-source nature of Linux also allows flexibility in customization and optimization for the specific needs of The Gaming Room.

**Operating Systems Architectures**

Linux’s modular architecture, coupled with lightweight containers like Docker, ensures scalability and reliability for handling multiple gaming sessions simultaneously. The architecture supports efficient process management, robust networking, and resource isolation, making it suitable for a high-demand application like Draw It or Lose It. Additionally, this architecture supports multiple CPU architectures and memory configurations, enabling flexibility across diverse server environments.

#### **Storage Management**

A relational database management system (RDBMS), such as MySQL or PostgreSQL, is recommended due to its ability to efficiently store and retrieve data while supporting unique identifiers to maintain data integrity. To address additional storage needs, the solution should account for storage beyond the database system itself. Incorporating supplementary storage options ensures that the system can manage increased data demands effectively. High availability and fault tolerance can be achieved through replication and clustering, while automated backup and restoration processes help safeguard against data loss.

#### **Memory Management**

Linux optimizes memory usage with techniques like demand paging, cache management, and virtual memory. Using the Singleton design pattern ensures only one instance of the game is in memory, avoiding redundancy and enhancing performance. Additionally, memory allocation tools available in Linux can help monitor and fine-tune resource utilization. The use of in-memory data caching services, like Redis, can further enhance performance by reducing database query times for frequently accessed data.

#### **Distributed Systems and Networks**

The application will employ RESTful APIs to facilitate platform communication, ensuring consistent interactions between client devices and the server. Distributed systems will rely on cloud-hosted servers for high availability and scalability. Load balancers will manage traffic effectively, while redundancy measures address potential connectivity issues. Connectivity dependencies between components will be addressed by implementing retries, failover mechanisms, and monitoring tools. Incorporating content delivery networks (CDNs) can also improve response times for users across different geographic locations.

**Security**

Protecting user information across platforms is paramount, and key measures not only meet but exceed the rubric's expectations for securing user data by implementing cutting-edge security technologies and processes. For instance, adopting end-to-end encryption ensures data remains confidential throughout its lifecycle, safeguarding it against interception or unauthorized access. Multifactor authentication (MFA) adds an additional layer of defense, significantly reducing the risk of unauthorized access by requiring multiple forms of verification. Regular penetration testing and vulnerability assessments are conducted to proactively identify and mitigate potential threats, showcasing a commitment to maintaining robust security practices. These measures are seamlessly integrated with the distributed system to ensure consistent and comprehensive protection for user data, regardless of the platform.

Critical security measures include securing communication, protecting data, employing robust authentication mechanisms, and ensuring network and cloud security. Secure communication is achieved by using HTTPS protocols for encrypted data exchanges. For data protection, database encryption is implemented to secure sensitive information. Authentication mechanisms such as OAuth or similar protocols are utilized to provide secure and reliable user authentication. Network security is bolstered through firewalls and intrusion detection systems, which guard against unauthorized access and attacks. Additionally, cloud security measures are employed, leveraging built-in features provided by platforms like AWS or Azure, including role-based access control (RBAC) and encryption at rest. Regular security audits are also conducted to proactively identify and resolve vulnerabilities, ensuring a continually fortified security framework.

**Conclusion**

By leveraging Linux as the operating platform, employing scalable storage and memory management solutions, and implementing robust distributed system and security measures, The Gaming Room can successfully expand *Draw It or Lose It* to a cross-platform solution that meets user expectations and achieves business objectives. These steps not only ensure a secure and reliable user experience but also position the platform as a trusted and innovative player in the gaming industry.