

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 | 11/21/24 | Jacob Wisniewski | Initial creation |
| 1.1 | 12/15/2024 | Jacob Wisniewski | Added recommendations |

**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, which is currently an Android-only application. The new version must serve multiple platforms, such as mobile, web, and desktop, and allow multiple teams to participate. The proposed solution is to create a scalable, distributed web application architecture, which will enable efficient team management and game sessions. We recommend using a cloud-based solution to facilitate a unified, multi-platform experience while allowing scalability for a larger user base. The solution will involve designing unique identifiers for teams, players, and game sessions to ensure data integrity and uniqueness throughout the system.

## Requirements

*The client has the following business and technical requirements:*

* The game must support one or more teams.
* Each team can have multiple players.
* Team and game names must be unique.
* Only one instance of the game must exist in memory at any given time.

## [Design Constraints](#_2et92p0)

The design of the game application must address various constraints imposed by the need for a web-based distributed environment. The main constraints include network latency, ensuring that the application performs well across platforms with varying network conditions. Data consistency and concurrency control are also critical, as multiple users might attempt to join, create teams, or interact with the game simultaneously. The application must implement proper synchronization techniques to avoid race conditions and ensure that unique identifiers are properly enforced. Additionally, security measures need to be integrated to protect user data and interactions across distributed systems.

## [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 UML class diagram represents the core components of the game application. It includes entities such as Game, Team, and Player, all of which inherit from a common base class named Entity. This base class is used to hold common attributes like unique identifiers. The Game class has a composition relationship with the Team class, meaning a Game can contain multiple teams. Each Team class has an aggregation relationship with the Player class, indicating that teams consist of multiple players. Object-oriented principles such as inheritance (Entity class) and composition/aggregation are utilized to ensure code reusability, flexibility, and clarity in relationships between classes. This model allows us to fulfill the requirements for unique names, the creation of multiple teams, and unique instances in memory.

"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** | macOS offers a stable and user-friendly environment for hosting web-based applications. However, server-side support might be limited compared to Linux. Performance is sufficient for moderate workloads. | Linux is ideal for hosting web applications due to its stability, scalability, and performance. It is also open-source, which reduces hosting costs. | Windows Server is easy to use and offers good support, especially for .NET applications. However, licensing costs may be higher. | Mobile devices are not well-suited for hosting. However, mobile clients are essential for supporting the user experience across devices. |
| **Client Side** | Developing for macOS requires knowledge of Xcode and Swift, which can increase the development cost and time for cross-platform compatibility. | Linux client-side development can vary depending on the Linux distribution. Generally, it requires more expertise to support all distributions. | Developing for Windows clients is straightforward and uses well-established tools such as Visual Studio. | Mobile development requires expertise in both Android (Kotlin/Java) and iOS (Swift). This increases time and costs for development and testing. |
| **Development Tools** | Relevant tools include Xcode, Swift, and JavaScript frameworks such as React for web clients. | Development tools include Eclipse, VS Code, and various web frameworks, such as Node.js and Angular. | Tools such as Visual Studio, .NET, and JavaScript frameworks are commonly used. | Tools include Android Studio, Xcode, React Native, and Flutter for cross-platform development. |

## 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**: The best operating platform for hosting "Draw It or Lose It" is Linux. It offers excellent scalability, stability, and performance for web applications, while being cost-effective due to being open-source. Using a cloud-based provider like AWS with Linux instances will allow easy deployment of containerized applications and microservices, meeting the client’s requirements for scalability and cross-platform compatibility.
2. **Operating Systems Architectures**: Linux’s architecture provides strong support for multitasking, virtual memory, and multi-user environments. Its modular kernel structure allows for customization to optimize server performance. Coupled with Docker for containerized applications, the Linux environment ensures portability, resource isolation, and minimal overhead, making it ideal for managing game sessions and teams in a distributed system.
3. **Storage Management**: The application should utilize cloud-based storage solutions like AWS S3 or DynamoDB for managing game state, player data, and team information. These systems are designed for high availability and scalability, ensuring data integrity across distributed platforms. DynamoDB, in particular, can handle concurrent updates with its atomic transactions, perfect for ensuring unique names and IDs.
4. **Memory Management**: Linux implements advanced memory management techniques such as paging, swapping, and demand paging, which optimize resource utilization. For the application, memory management can be enhanced by employing in-memory caching tools like Redis to store frequently accessed data, such as game session states and leaderboards, to reduce latency and improve response times.
5. **Distributed Systems and Networks**: A microservice architecture should be implemented for distributed functionality. Each microservice can communicate over APIs, ensuring modularity. For service-to-service communication, gRPC is recommended. gRPC allows efficient, low-latency communication, and uses Protocol Buffers for serialization of messages. For network interconnectivity, using secure HTTPS protocols and WebSocket for client-facing interactions will ensure low-latency communication between devices. Load balancers and redundancy mechanisms should be added to handle connectivity issues or outages.
6. **Security**: To protect user data and interactions, implement HTTPS for encrypted data transfer. Sensitive information like user credentials should be stored using encryption techniques such as AES and hashed using algorithms like bcrypt. Implement role-based access controls to restrict data access. Additionally, use secure OAuth 2.0 protocols for authentication and employ tools like AWS Web Application Firewall to safeguard against threats such as SQL injection or DDoS attacks.