

<Project 2 >

# **CS 230 Project 2 Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 2.0 | <09/18/24> | <Quinton Price> | < modernize their popular game > |

**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 is embarking on a project to modernize their flagship game, Draw It or Lose It, by creating a web-based version that supports real-time multiplayer functionality across multiple platforms. The main objective is to ensure scalability and the ability to host thousands of players at once, while maintaining only one instance of the game in memory. Using object-oriented principles, the project will implement the singleton pattern to manage game instances and the iterator pattern to ensure unique names for games, teams, and players. This distributed architecture will enable seamless, synchronized gameplay on various platforms, including desktops, laptops, and mobile devices.>

## Requirements

< The solution must fulfill the following business and technical requirements:

Unique Game Instance: Ensure that only one instance of the game can exist at any given time, and each game must have a unique name.

Progressive Rendering: During gameplay, images must be progressively revealed over a one-minute time span, with guessing opportunities for each team.

Scalability: The system must handle thousands of players across multiple platforms while maintaining real-time performance.

Cross-Platform Compatibility: The system must support web browsers on Mac, Linux, Windows, and mobile platforms like iOS and Android.

Security: The game must implement secure authentication mechanisms and ensure encrypted communication between clients and the server.>

## [Design Constraints](#_2et92p0)

< When building a distributed, web-based game application, the following constraints must be considered:

Scalability: The architecture must efficiently manage thousands of concurrent users without affecting performance.

Real-time Communication: Due to the game’s real-time nature, it must maintain synchronized drawing and guessing interactions between all players.

Cross-platform Compatibility: The system must be compatible with a wide range of devices, ensuring a responsive user interface regardless of whether the player is on a desktop, tablet, or mobile device.

Concurrency Management: Handling multiple simultaneous connections requires robust state and resource management to ensure players receive timely updates, which can affect the overall architecture.>

## [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 showcases the object relationships within the game application. The Entity class is the base class from which Game, Team, and Player inherit. This demonstrates the use of inheritance to reduce redundancy and encapsulate shared properties like id and name. The use of design patterns is evident in the singleton pattern for GameService, ensuring only one instance of the game exists. The iterator pattern ensures that games, teams, and players can be added only if their names are unique, thus fulfilling the client’s requirement for unique identifiers.>

**"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)

For each platform (Linux, Mac, Windows, and Mobile Devices), you can evaluate:

Server Side: Linux is recommended for its scalability and stability, though all platforms can host web applications.

Client Side: Each platform has specific browsers to support (Safari for Mac, Edge for Windows, etc.). Mobile development focuses on responsive design.

Development Tools: Xcode for Mac, Android Studio for mobile, Eclipse for Windows/Linux—ensure the development team's tools align with the platform.

| **Development Requirements** | **Mac Xcode, IntelliJ IDEA, and JavaScript development are common. Costs can be high due to limited developer availability.** | **Linux Eclipse, IntelliJ IDEA, and JavaScript development tools are widely available and cost-effective on Linux.** | **Windows Eclipse and Visual Studio are the primary development tools, with extensive support for Java, JavaScript, and .NET.** | **Mobile Devices**  **Development requires tools like Android Studio and Xcode, with frameworks like Bootstrap for responsive design. Testing is necessary across multiple devices.** |
| --- | --- | --- | --- | --- |
| **Server Side** | <Mac is less common for web hosting but can support web applications. Development and testing costs may be higher due to fewer available developers.> | < Linux is the preferred platform for web hosting due to its scalability, open-source nature, and stability. It offers the most cost-effective and flexible server-side solution.> | < Windows servers can host web applications, but licensing costs are higher. However, it offers robust support and development tools for enterprise environments.> | < Mobile hosting is not typically needed for this type of application, but mobile compatibility must be ensured for the front-end.> |
| **Client Side** | < Mac clients need compatibility with Safari and other web browsers, which may require additional testing.> | < Linux users typically use Chrome or Firefox for web applications, and testing on these platforms is critical.> | < Windows dominates the desktop market, so ensuring compatibility with Edge, Chrome, and Firefox is essential.> | < Responsive web design and touch-friendly interfaces are necessary for iOS and Android users to ensure smooth gameplay.> |
| **Development Tools** | < Xcode, IntelliJ IDEA, and other IDEs work well on Mac. JavaScript and Java are common languages for web-based applications.> | < wide range of development tools such as Eclipse, IntelliJ IDEA, and NetBeans, and can run server software seamlessly. Java and JavaScript are common for web development.> | < Eclipse, and IntelliJ IDEA are key tools for development on Windows. The platform is compatible with Java, JavaScript, and .NET technologies.> | < development requires tools like Android Studio for Android devices and Xcode for iOS. Responsive web design frameworks like Bootstrap are also essential for mobile compatibility.> |

## 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**: <I recommend Linux as the primary platform for hosting Draw It or Lose It. Linux provides exceptional scalability, stability, and flexibility, making it an ideal choice for hosting the game’s distributed architecture. Additionally, it’s cost-effective, with no licensing fees, and widely used in the web hosting industry.>
2. **Operating Systems Architectures**: < Linux’s Unix-like architecture makes it highly reliable for distributed systems. It supports both vertical and horizontal scaling, meaning that as the number of users increases, the system can adapt without significant changes to the architecture.>
3. **Storage Management**: < A distributed database such as MongoDB or MySQL is recommended to manage game states, player data, and in-game drawings. These systems are optimized for handling large volumes of data and are capable of supporting multiple clients simultaneously.>

**Memory Management**:

1. **Distributed Systems and Networks**: < Linux’s efficient memory management, including features like paging and demand-based allocation, makes it well-suited to manage high-load scenarios where many players are interacting with the system at the same time. This ensures that the server can handle the game's real-time nature.>

**Security**: < The application should enforce SSL/TLS encryption for all communications to protect data integrity. For authentication, OAuth 2.0 is a widely accepted framework that ensures secure user access. Passwords and sensitive data should be hashed using algorithms like bcrypt to prevent unauthorized access.>