   
 <Name-of-Software-Application>

**CS 230 Project Software Design Template**

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 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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, a client of Creative Technology Solutions, has requested a web-based version of their Android game, Draw it Or Lose it. The game is a team-based competition where players guess what is being drawn on the screen over seven-time rounds

This document presents a software solution that uses object-oriented programming and industry-standard design patterns to meet the clients’ needs. The singleton pattern will be implemented to ensure only one game instance exists in memory at a time, while the iterator pattern will manage collections of teams and players and enforce unique names. This approach provides a scalable, maintainable foundation for expanding the game across multiple platforms

Requirements

The Gaming Room’s web-based game must support multiple teams, each with multiple players. Game and team names must be unique. Only one game instance should exist in memory at any time. Each game team, and player must have a unique ID. The application should follow object-oriented design, use the Singleton and iterator patterns, and work across multiple platforms.

[Design Constraints](#_2et92p0)

Because this is a web-based game, the application must be bult to run in a distributed environment. This limits the use of shared memory and requires careful control over data flow between clients and servers. Only one game instance can exist at a time, which is handled using the Singleton pattern. Name uniqueness must be enforced programmatically using iteration through collections. The applications must also be platform-independent and scalable, meaning it must run on various devices and support future growth without major redesigns.

[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 Diagram for the game application defines four main classes: Game, Team, Player, and Entity. The Entity class acts as the base class that contains shared properties such as ID and name. the Game, Team, and Player classes inherit from Entity, which demonstrates the object-oriented principle of inheritance. Each class is responsible for managing its own data, showing encapsulation by keeping attributes private and exposing functionality through public methods. The GameService class is used to manage the creation and storage of games, teams and players, enforcing the Singleton pattern to allow only one active game instance in memory. The iterator pattern is used to ensure name uniqueness when adding new teams or players.

**"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** | macOS is stable, but it is not widely used for hosting web-based applications in production. Limited server deployment tools compared to Linux makes it less ideal for backend hosting | Linux is a preferred platform for hosting web-based apps. It offers strong security open-source tools, and great performance for server-side development and deployment | Windows servers are compatible with many enterprise tools but are less common for hosting java-based web apps. Licensing costs and system resource usage can be drawbacks | Mobile devices are not used for hosting web applications due to limited processing power, memory, and lack of persistent internet connectivity. |
| **Client Side** | Developing for macOS requires specific hardware and knowledge of Apple development environments. Cross-Platform testing increases time and cost. | Linux is less common as a client platform but still supports modern browser’s and java. Developing for the Linux requires experience with open-source tools and compatibility testing | Windows is the most common desktop OS, making it a priority for testing and support. It offers broad compatibility and widely used development | Development for mobile requires responsive design and testing on both IOS and Android. It adds time and complexity, but it is essential for user accessibility. |
| **Development Tools** | Tools like IntelliJ IDEA, Eclipse and Xcode are available. Hava development is supported, but Xcode is primarily used for Apple-specific apps. | Supports Eclipse, IntelliJ, and open-source command-line tools. It’s a favorite for backend development and offers flexibility for web-based java apps | Offers a full range of development tools including Eclipse, IntelliJ, and Visual Studio. Its user-friendly and widely supported. | Development requires Android Studio for Android and Xcode for IOS, Mobile apps must be tested on emulators or real devices during 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**: I recommend deploying the game on a Linux-based platform. It’s cost-effective, reliable, secure, and highly scalable for web-based application

1. **Operating Systems Architectures**: The application should follow a client-server architecture, where the game logic and database reside on a centralized Linux server, and user connect through web browsers on various client devices
2. **Storage Management:** A relational database such as MySQL or PostgreSQL is ideal for managing game, team, and player data. These systems offer fast queries, support for relationships and can be scaled for growth.
3. **Memory Management**: Linux efficiently manages memory through virtual memory techniques and swap space. This ensures that even under load, the single-instance game can run smoothly and handle multiple user sessions
4. **Distributed Systems and Networks: The game should use a RESTful API over HTTP/HTTPS to allow communication between clients and servers. This enables multiple platforms to interact with the same backend while handling disconnects and retries if network issue occurs**
5. **Security** User data should be secured through HTTPS encryption and input validation to prevent injection attacks. Authentication should be implemented using token-based systems like JWT, and sensitive user information should be stored with encryption at rest

Version 1.0