

<Draw it or loose 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 | <07/16/2023> | <Azaria Berry> | < Updated the code by adding a new class to to maintain only one instance of the game in memory at any given time. This can be achieved by assigning unique identifiers to each game, team, and player instance.> |

**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 requested assistance in developing a web-based version of their gaming app. This software design document aims to outline the solution for streamlining the development process and meeting the client’s software requirements.

The game application will allow multiple teams to participate, with each team consisting of several players. To ensure uniqueness and prevent conflicts, game and team names will be required to be unique. Additionally, it is crucial to maintain only one instance of the game in memory at any given time. This can be achieved by assigning unique identifiers to each game, team, and player instance.

## Requirements

*Client's Business Requirements:*

*1. Ability to have one or more teams in a game.*

*2. Multiple players should be assigned to each team.*

*3. Game and team names must be unique to allow users to check availability.*

*4. Only one instance of the game should exist in memory at any given time.*

*Client's Technical Requirements:*

*1. Web-based development for the gaming app.*

*2. Streamlined development process.*

*3. Unique identifiers for game, team, and player instances.*

## [Design Constraints](#_2et92p0)

Design Constraints for Developing the Game Application in a Web-Based Distributed Environment:

1. Web-Based Distributed Environment:

- Implication: The game application will operate over the internet and be distributed across multiple servers or nodes.

- Implication: Network communication needs to be efficient to ensure smooth gameplay.

- Implication: Scalability considerations must be taken into account to handle a large number of concurrent users.

- Implication: Robust security measures should be implemented to protect data and ensure secure communication.

2. Unique Game and Team Names:

- Implication: The game and team names must be validated for uniqueness.

- Implication: The system should provide immediate feedback to users when selecting names to indicate availability or conflicts.

3. Single Instance of the Game:

- Implication: Each game, team, and player instance needs to be assigned a unique identifier.

- Implication: Proper resource management is necessary to handle the state of the game and prevent conflicts or inconsistencies.

Addressing these design constraints will require careful consideration of network communication, scalability, security measures, name validation, user feedback, and resource management techniques to ensure the fun and safety of the user.

## [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)

* Entity Class:

This is an abstract base class that provides common attributes and behaviors for entities in the application.

It has two attributes: id (a long) and name (a String).

It has a constructor that takes an identifier and a name to initialize the attributes.

It provides getter methods for id and name.

This class demonstrates the Object-Oriented Programming (OOP) principle of abstraction by providing a common structure for other entities to inherit from.

* Game Class:

This class inherits from the Entity class, as represented by the arrow with an open triangle head from Game to Entity.

It has a constructor that requires an identifier and a name to create a new game instance.

The Game class demonstrates the OOP principle of inheritance, as it inherits common attributes and behaviors from the Entity class.

* Player Class:

This class also inherits from the Entity class, as represented by the arrow with an open triangle head from Player to Entity.

It has a constructor that takes an identifier and a name to create a new player instance.

The Player class demonstrates the OOP principle of inheritance, as it inherits common attributes and behaviors from the Entity class.

* Team Class:

Similar to the Game and Player classes, the Team class also inherits from the Entity class.

It has a constructor that requires an identifier and a name to create a new team instance.

The Team class demonstrates the OOP principle of inheritance, as it inherits common attributes and behaviors from the Entity class.

* GameService Class:

This class represents a singleton service for the game engine, as shown by the stereotype <<singleton>>.

It maintains a list of active games using the games attribute (List<Game>).

The nextGameId attribute keeps track of the next game identifier to be assigned.

The instance attribute holds the reference to the singleton instance of the GameService class.

The GameService class demonstrates the OOP principle of encapsulation by encapsulating the list of games and providing access through its methods.

* SingletonTester Class:

This class is used to test the behavior of the singleton pattern in the GameService class.

The testSingleton() method demonstrates how the singleton instance of GameService can be obtained and used to access the list of games.

* ProgramDriver Class:

This class contains the main() method, which serves as the entry point for the application.

It demonstrates how the GameService instance can be accessed using the singleton pattern and how games can be added and retrieved from the service.

Relationships:

The GameService class has an association relationship with the Game, Player, and Team classes. This relationship indicates that the GameService class uses instances of Game, Player, and Team classes in its operations.

The Game class has a generalization relationship with the Entity class. This relationship represents inheritance, indicating that the Game class inherits attributes and behaviors from the Entity class.

Similarly, the Player and Team classes also have a generalization relationship with the Entity class.

**"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** | Mac offers a robust and stable environment for hosting web-based software applications. Its Unix-based architecture ensures reliability and security, making it a suitable choice for server deployments. Advantages of using Mac for server-side hosting include its native support for common web development tools and technologies. However, Mac may have higher initial hardware costs compared to Linux and Windows. Additionally, the client should consider the potential licensing costs for macOS Server if using advanced features. | Linux is well-known for its scalability, performance, and cost-effectiveness, making it a preferred choice for server-side hosting. It provides a wide range of server deployment options, including popular web servers like Apache and Nginx. Linux’s open-source nature ensures a vast community and continuous development, leading to frequent updates and enhanced security. The cost-effectiveness of Linux is particularly advantageous for The Gaming Room’s distributed environment. | Windows offers a user-friendly and familiar environment for server-side hosting, making it accessible to developers with a background in Windows development. It provides robust support for Microsoft technologies and server applications, which can be advantageous for certain applications. However, licensing costs for Windows Server and related software should be carefully considered, as they can be higher than Linux counterparts. | Mobile devices vary in characteristics based on the operating system (e.g., iOS, Android). While they are not typically used for server-side hosting in the traditional sense, they play a crucial role in accessing the web-based software application as clients. Ensuring compatibility with mobile devices requires responsive web design and optimization for different screen sizes and device capabilities. Cloud-based server solutions can handle server-side hosting while serving mobile clients efficiently. |
| **Client Side** | Developing for Mac as a client-side platform requires expertise in Apple’s development ecosystem, including Xcode as the primary Integrated Development Environment (IDE). Mac development can be more cost-intensive due to the need for Apple hardware and software licenses. Time considerations depend on the developers’ familiarity with Apple technologies, and expertise in Swift and Objective-C is necessary for iOS-specific functionality. | Developing for Linux as a client-side platform demands proficiency in web technologies, including HTML, CSS, and JavaScript, as well as cross-platform compatibility. The cost of development for Linux is generally lower due to open-source tools and the availability of free IDEs like Visual Studio Code. The development process can be efficient with experienced web developers, and the time required depends on the complexity of the user interface and interactions. | Developing for Windows as a client-side platform involves using languages such as C#, JavaScript, and C++ and leveraging Microsoft’s development tools, primarily Visual Studio. The cost of development for Windows can vary based on the chosen tools and the developers’ expertise in Microsoft technologies. Time considerations include learning curve and integration with Windows-specific features. | Developing for mobile devices involves supporting multiple platforms, such as iOS and Android. This requires expertise in platform-specific languages (Swift and Objective-C for iOS, Java and Kotlin for Android) and responsive web design for web-based interfaces. The cost and time considerations depend on the choice between native app development and responsive web design. Cross-platform development tools like React Native and Flutter offer cost-effective alternatives for supporting both platforms with a single codebase. |
| **Development Tools** | Mac development commonly involves using Xcode, Apple’s official IDE, which supports Swift and Objective-C for iOS and macOS development. Additional tools like CocoaPods and SwiftUI are available for enhancing development productivity and user interface design. Licensing costs are associated with Apple Developer Program enrollment and macOS Server for advanced features. | Linux development mainly revolves around open-source tools, including Visual Studio Code as a popular choice for web development. Web developers can utilize standard web technologies like HTML, CSS, and JavaScript. The open-source nature of development tools contributes to cost-effectiveness, as most of them are freely available. | Windows development centers on Microsoft’s Visual Studio IDE, providing extensive support for languages like C#, JavaScript, and C++. Microsoft offers various editions of Visual Studio, with licensing costs varying based on the chosen version and features required. The cost of development tools for Windows can be higher compared to Linux’s open-source tools. | Mobile development depends on the platform, with Xcode as the primary tool for iOS app development (Swift and Objective-C), and Android Studio for Android app development (Java and Kotlin). Licensing costs for mobile development tools are associated with Apple Developer Program enrollment and Google Play Console for app distribution. Cross-platform development tools like React Native and Flutter offer cost-effective alternatives for supporting both platforms with a single codebase. |

## 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**: To facilitate the expansion of Draw It or Lose It, I recommend adopting a cloud-based operating platform such as Amazon Web Services (AWS). AWS offers a wide range of services, including virtual servers (EC2), serverless computing (Lambda), and database solutions (RDS), which can accommodate the game's growth across different computing environments.
2. **Operating Systems Architectures**: AWS supports various operating system architectures, including Windows, Linux, and UNIX. This flexibility allows the client to develop and deploy the game on different operating systems while maintaining consistent performance and functionality.
3. **Storage Management**: For storage management, AWS provides Amazon S3 (Simple Storage Service) which offers scalable and durable object storage. It allows the client to store game assets, user data, and configurations securely, with the option for versioning and replication across multiple regions.
4. **Memory Management**: AWS's Elastic Compute Cloud (EC2) instances utilize hypervisor-based virtualization for memory management. This ensures that memory is efficiently allocated to instances based on their needs, preventing resource wastage and optimizing overall performance for Draw It or Lose It.
5. **Distributed Systems and Networks**: To achieve cross-platform communication, AWS provides services like Amazon API Gateway for creating APIs that enable communication between devices and platforms. Distributed systems can be designed using micro-service architecture, where each component of the game operates as a separate service that communicates over APIs. Redundancy, load balancing, and auto-scaling mechanisms can be employed to mitigate connectivity issues and outages, ensuring uninterrupted gameplay.
6. **Security**: AWS offers robust security features, including Identity and Access Management (IAM) for user authentication and authorization. The game's data can be encrypted both at rest and in transit using AWS Key Management Service (KMS) and Transport Layer Security (TLS). Compliance with regulations such as GDPR can be achieved by utilizing AWS's compliance certifications and tools.