

<Draw It or Lose It V.1.0>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

**[CS 230 Project Software Design Template](#_Toc115077317)** [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 | <02/15/2024> | <Eli Rubin-Calvert> | <final version of Project 2> |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

A game needs to be created that has only one instance in memory at a time. To accomplish this, I recommend implementing a singleton pattern, to ensure that unique identifiers for each instance are created in an organized manner. Names in the game should be required to be unique to avoid any duplicates or data errors. A database needs to be maintained to keep data updated, and logic needs to be created to have the ability to have multiple teams involved concurrently, and multiple players on each team needing fast updates to any actions taken in the game that effect any other character.

## Requirements

A game will have the ability to have one or more teams involved.

Each team will have multiple players assigned to it.

Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.

Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

In a distributed environment, network latency can be a issue. The delay in transmission between the client and server can affect responsiveness. As an implication of this, the developers should optimize the game's architecture and implement techniques like client-side rendering to reduce the impact of latency. games may experience variable loads and user numbers, and the game must be able to scale to handle a large number of concurrent users. Developers should use tools such as cloud solutions and load balancer’s to reduce the strain on the system. Security also needs to be considered as ,many games have issues related to security and hackers. Security measures, such as data encryption, user authentication, and authorization need to be implemented by the developers.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

Entity is a superclass for Game, Player, and Team classes. This means that Game, Player, and Team have their attributes and methods inherited by the Entity class. Game inherits from the GameService class. Entity has Attributes: Id (long), name (String) and Methods: getld(), getName(), toString().

GameService class has Attributes: games (List<Game>), nextGameld (long), nextPlayerld (long), nextTeamld (long), service (GameService) Methods: GameService(), getInstance(), addGame(name: String), getGame(id: long), getGame(name: String), getGameCount(), getNextPlayerld(), getNextTeamId(). GameService has a composition relationship with Game since it contains a List of Game objects. It implements the Singleton design pattern, ensuring that there is only one instance of GameService throughout the application. Object oriented principles are seen in the classes also.

**"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 has many advanced developer tools to facilitate faster development. Mac has many security features to help provide security for apps, but is lesser used for a Server OS so it may be less optimized for certain tasks. Some additional costs may be needed if using Mac, due to the fee Apple takes, and potential costs related to maintenance. | Linux us open source and has a variety of distros available that can offer additional flexibility. It is highly secure and stable and is free. It also has a large developer community. It has a lot of built on optimizations making it a strong choice. Linux has more advantages related to licensing costs due to its open source nature. | Many applications have used this OS for a very long time, leading to a stable product that has many tools and integrations available. There have been some security issues facing this OS, but the long history of using this OS makes it a solid choice. | They generally have less resources and rely more on unstable connections leading to issues hosting games. There can also be platform specific challenges due to the large amount of device types. Windows offers mature options for deployment and hosting, but can come at a higher cost due to licensing fees. |
| **Client Side** | Mac development may need additional hardware for testing purposes. Experts in Mac development and design are needed to support a variety of clients. These constrains can lead to higher costs, but also potentially greater efficiency. | Linux development has a lot of cost savings due to its open source nature, but the time to understand the associated tools and libraries and API’s also needs to be considered. There is a large community that can help reduce this issue, but it could lead to additional development time being needed compared to alternatives. | Development for this OS has much the same considerations as Mac. Various types of devices need to be supported, and specific dev tools and API’s need to be learned by the developers. There are many libraries and existing tools that can be leveraged to help develop the client side from the many years the ecosystem has existed. | For mobile development additional factors need to be considered such as if the development needs to be cross platform. If so, additional devices may be needed for testing. Additional expertise of specific device developers may also be needed depending on the requirements of the app. |
| **Development Tools** | Swift is Apple's primary programming language for macOS. Objective-C is outdated but also used in some cases. Xcode is Apple's official integrated development environment for macOS | C and C++ are commonly used for system-level programming and building applications on Linux. Python and other languages can also be used. Linux has specific libraries, but can use any IDE to build the game. | Java is commonly used for this OS, and IDE’s such as Visual Studio, but any IDE or programming language can be used. | Swift, React Native, Flutter, and Kotlin are all commonly used for mobile development. IDE’s built for this include Xcode and Android Studio. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>

A web based system should be implemented to enable the game to expand to other computing environments. Mobile devices and desktop can both be covered by this choice. Linux has the most flexibility and the least cost so that is my recommendation. Linux also has been developed efficiently and had good code standards and is tested extensively, so the risk is not as high as some alternatives in this area.

1. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>

The software will run on top of web servers hosted on Linux. Linux is a Unix-like operating system that follows a modular kernel architecture. The kernel is responsible for managing hardware resources, providing system services, and acting as an facilitator between hardware and user-level applications. Above the kernel, Linux has a user space where applications and user-level processes run. Linux provides a solid foundation for hosting web servers and running web applications, due in part to its modular kernel architecture and extensive open-source ecosystem.

1. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>

A Relational Database Management System should be used such as PostgreSQL. Postgres is open source and reliable. Postgres can manage structured data for user accounts and offers data consistency, scalability, and reliability. Its transactional nature and support for data consistency ensure that critical data operations are executed reliably, even in the event of system failures or errors.

1. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>

In a web-based platform, memory management is handled by the web server, application server, and the client's device. The client-side manages its memory efficiently, while server-side memory management depends on the selected environment and programming language. Using both client-side and server-side techniques tailored to the specific requirements of the Linux based platform is my recommendation for the biggest impact.

1. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>

To enable Draw It or Lose It to communicate between various platforms, a RESTful API can be implemented. The API allows different components, such as the game server, client applications, and user accounts, to interact. Connectivity for the components must be maintained and secured. In case of network connectivity issues or outages, error handling and offline functionality on the client side should be implemented. Having good documentation of the API design and infrastructure of the game is also important to ensure the game can be maintained easier.

1. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.> Security measures, including HTTPS, encryption, authentication, and authorization should be developed. Existing security features of linux should be leveraged. HTTPS ensures data encryption in transit. User authentication and authorization mechanisms should be implemented to protect user accounts and data from unauthorized access. Using third paty tools such as OAuth 2.0 for secure integration's is also recommended. Hiring third party testers and tools is another thing that the game should implement to improve security.