

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 | 07/10/23 | John Nguyen | First Submission |
| 1.1 | 07/24/23 | John Nguyen | Second Submission |
| 1.2 | 08/7/23 | John Nguyen | Third Submission |

**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 asked Creative Technology Solutions(CTS) to develop a web-based version of their Android app game, Draw It or Lose It.

## Requirements

* The game must be accessible from a web browser.
* 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.

## [Design Constraints](#_2et92p0)

The game must be able to run on most internet browsers and it must account for different screen sizes.

## [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 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.**

## 

The program utilizes a central driver class to initiate the creation of games, teams, and players. The GameService class manages the creation process and follows the singleton design pattern to ensure only one instance exists in memory. To accomplish this, GameService's constructor method is private, and the getInstance() method checks for the presence of an instance before starting.

Once GameService is established, the driver class can use the addGame() method to create a new game, the addTeam() method to add a team, and the addPlayer() method to add a player. These 3 methods use the iterator pattern to prevent duplicate names of their respective object.

All classes, such as Game, Team, and Player, inherit from the Entity class, which includes protected attributes like id and name. This shows the object-oriented technique of inheritance and modularity. By having the subclasses inherit properties, it helps keep the code modular and DRY(do not repeat yourself). The Entity constructor is protected to prevent the creation of null objects.

## [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** | According to Apple’s [website](https://support.apple.com/en-us/HT208312), Mac OS Server has been discontinued as of 4/21/22. Therefore, using a Linux Server would be the next solution. | This OS is stable, secure, customizable, and scalable. It has strong command-line and scripting capabilities and a supportive community. Some disadvantages are that it requires advanced technical knowledge and has a steep learning curve for beginners.  Since most Linux distros are open-sourced and free, costs will be cheaper when using Linux servers. | Windows Server is a reliable and secure option for server environments. It has native support for services and features like Active Directory and .NET framework. It integrates well with the Microsoft ecosystem and offers support for Windows-specific technologies.  However, there are licensing costs and higher resource requirements. In addition, cross-platform compatibility may be limited for certain web applications. | It is not recommended to use mobile device operating systems as servers. |
| **Client Side** | Since we will be building a web application that will run in the browser, client-side requirements will be nearly identical for all traditional desktop OSes.  To ensure a web app works on any device, prioritize responsive design, verify cross-browser compatibility, optimize for mobile devices, account for browser limitations, and thoroughly test across platforms. | Since we will be building a web application that will run in the browser, client-side requirements will be nearly identical for all traditional desktop OSes. See Mac column. | Since we will be building a web application that will run in the browser, client-side requirements will be nearly identical for all traditional desktop OSes. See Mac column. | The mobile devices will follow most of the same processes of the traditional desktop OSes. The major differences are that we would need to account for the smaller screen sizes and touch-based input. |
| **Development Tools** | Desktop OSes will have similar requirements.  Front-end Programming Languages: HTML, CSS, JavaScript  Back-end Programming Languages: Depends on what is decided by the development team. It could be Java, NodeJs, or something else.  IDEs: Xcode appears to be a common choice on Mac OS. However, popular IDEs such as Sublime Text, Atom, and Visual Studio Code are compatible with Mac, Linux, and Windows OSes.    There will probably be multiple development teams, such as a team for the front end and a team for the back end. | Desktop OSes will have similar requirements. See Mac requirements. | Desktop OSes will have similar requirements. See Mac requirements.  Visual Studio is a powerful IDE exclusive to Windows. However, any of the popular IDEs can also be used. | Since the app will be intended to be run from the browser, the front-end and back-end programming languages will be the same as the desktop OSes.  However, if the app were to be extended to become mobile apps on the respective OSes, then we would need some more languages. Java and Kotlin are used for Android development while Swift is used for iOS development.  Android Studio is a common IDE for Android development and Xcode is an IDE for iOS development. Android studio is free. Xcode itself is free, but there appears to be an optional subscription called Xcode Cloud. The price varies on the subscription level you sign up for.  I think the front-end and back-end team would be sufficient if only developing for the browser. However, if extended to mobile apps, then a dedicated mobile team may be necessary. |

## 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**: By utilizing Linux-based servers, we can reduce expenses in various ways. Firstly, Linux is an open-source platform, which means there are no upfront expenses or licensing fees associated with proprietary operating systems like Windows Server. Additionally, Linux is a lightweight system that can operate efficiently on less powerful hardware, enabling cost-effective server configurations without compromising performance. Lastly, Linux is widely supported by hosting providers and cloud platforms, offering cost-effective alternatives that are specifically designed for Linux-based deployments.
2. **Operating Systems Architectures**: Draw It or Lose It is a browser-based game that can be played on any operating system through web browsers. It uses a client-server architecture with a web application approach, with the front end built using web technologies and the back end consisting of a server that handles game logic and data storage. The server exposes a RESTful API for communication between the front-end and back-end, and the front-end is developed using responsive web design for compatibility across different devices.
3. **Storage Management**: Object storage is a perfect choice for managing and storing large volumes of unstructured data, such as the stock drawings and game data used in Draw It or Lose It. It's scalable, durable, and flexible, supporting various data formats. Cloud platforms like AWS, Azure, and GCP offer secure and reliable object storage services, seamlessly integrated with the recommended cloud-based operating platform. By utilizing an object storage system, The Gaming Room can efficiently manage and store the required content, ensuring efficient access, durability, and scalability.
4. **Memory Management**: JavaScript is used on the front end and it automatically manages memory allocation and deallocation. It utilizes techniques like automatic garbage collection to identify and free memory that is no longer needed, reducing the risk of memory leaks and optimizing memory usage.

JavaScript in the web browser handles memory allocation. It dynamically assigns memory for variables, arrays, and objects as needed during the execution of the game. The browser's memory management system keeps track of memory availability and ensures efficient allocation of memory resources.

1. **Distributed Systems and Networks**: When designing a system for a game, a microservices architecture may be used. This approach breaks down various components of the game into smaller, independent services that each handle specific functionalities such as game logic, user authentication, image rendering, and data storage. Using this modular approach makes the system more scalable, flexible, and easier to maintain. Communication between services can be achieved through the REST API calls.

To ensure optimal performance and handle increased traffic, load balancing techniques can be implemented. Load balancers distribute incoming requests across multiple instances of a service, preventing any single component from being overloaded. Auto-scaling mechanisms can also be used to automatically adjust the number of instances based on demand, allowing the system to handle varying loads efficiently. This approach ensures high availability, improves response times, and avoids single points of failure.

In order to maintain uninterrupted gameplay, it is crucial to build fault tolerance and resilience into the distributed system. Replicating data across multiple nodes ensures redundancy and availability. In case of failures, automatic failover mechanisms or replication to standby nodes can be used. By designing the system with fault tolerance in mind, the game can continue to function even if individual components or network connections experience failures.

1. **Security**: To ensure the security and protection of user accounts and data, it is essential to implement a secure user authentication mechanism. Authentication protocols like OAuth or JWT can effectively manage and verify user credentials. This significantly reduces the risk of unauthorized access and potential attacks.

It is also crucial to secure user data during transmission and storage. Secure protocols such as HTTPS should be used to encrypt data in transit. This ensures that sensitive information like user profiles, game progress, and communication remains confidential and protected from interception or tampering.

Additionally, implementing a comprehensive access control and authorization system is necessary to manage user permissions effectively. Role-based access control (RBAC) should be used and this allows fine-grained control over user privileges. This would ensure that only authorized users can perform specific actions or access sensitive functionalities and data, which ultimately enhances overall security.