

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 | 07/20/2024 | Andrew Racic | Document creation. Complete the executive Summary, design constraints, and domain model. |
| 1.1 | 08/01/2024 | Andrew Racic | Added information to the Evaluation table for evaluations of characteristics, advantages, and disadvantages of hosting a web-based game, as well as server side, client side, developmental tools, as well as determining what is required to host the application on multiple platforms. |
| 1.2 | 08/16/2024 | Andrew Racic | Added information on Recommendations for the client. |

**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 CTS to help with expanding their popular Android game, “Draw It or Lose It,” into a web-based application that is accessible across multiple platforms. The new version will retain the essence of the original game, where teams compete to guess puzzles based on a sequence of drawings. To meet this goal, Creative Technology Solutions (CTS) will develop a robust, scalable, and engaging web application that supports multiple teams and players, ensures unique naming conventions, and manages game instances effectively via a singleton pattern creation. Our solution will leverage modern web technologies to create a seamless gaming experience that is both fun and intuitive for users worldwide.

## Requirements

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

## Each team will have multiple players assigned to it.

## The 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)

 The primary design constraints for developing “Draw It or Lose It” in a web-based environment include:

* **Cross-Platform Compatibility:** The game must function seamlessly across various browsers and devices.
* **Concurrency Control:** Ensuring that only one instance of the game exists in memory at any given time requires a sophisticated session management system.
* **Unique Identifiers:** Implementing a mechanism to check for unique game and team names to prevent conflicts.
* **Performance:** Optimizing the rendering of drawings to maintain a steady and responsive user experience.
* **Scalability:** The system must handle a potentially large number of concurrent games without degradation in performance.

These constraints will guide the architectural decisions, such as the choice of a backend framework that supports real-time communication (e.g., WebSocket) and a frontend framework that is responsive and adaptive to different screen sizes and input methods.

## [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 game will be developed following the principles of OOP. Below is the UML diagram that depicts the design of the main classes in the game.

**"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.**

**Domain Model Description:** The UML diagram outlines the structure of the game application, detailing the classes and their relationships. The main classes include ProgramDriver, SingletonTester, GameService, Game, Team, and Player.

* **Program Driver**: This class serves as the main() driver class that will be used to initiate the creation of the games, players, and teams as well as the main entry point of the application.
* **SingletonTester**: This class might be used to test the Singleton design pattern, ensuring that only one instance of a class is created throughout the application.
* **GameService**: Acts as a service layer managing game instances, with methods to create games and manage unique identifiers. The GameService class follows our singleton design pattern so that only a single instance of GameService class can exist at any given time in the memory. Also the only way to initiate a GameService is through the getInstace() method. The getInstance method checks if a GameService has been started and will only start a new game as long as there is not currently one already in the memory.
* **Game**: Represents an individual game, with methods to add teams and players, reflecting the requirement that each game can have multiple teams.
* **Team**: Represents a team within a game, with the ability to add players, ensuring that each team has multiple players.
* **Player**: Represents an individual player, with attributes to identify them uniquely within a team.

**Object-Oriented Principles:**

* **Encapsulation**: Each class encapsulates its data and behavior, with private attributes and public methods.
* **Inheritance**: The Entity class, is a base class for common attributes and behaviors, suggesting that Game, Team, and Player inherit from it.
* **Association**: The diagram shows associations between GameService and Game, Game and Team, and Team and Player, indicating how these entities interact within the application.
* **Singleton Pattern**: A Singleton Pattern is a software design pattern that restricts the instantiation of a class to a singular instance. This means that only one instance of a class can exists in the entire program. It is one of the well known “Gang of Four” design patterns which describes how to solve recurring problems in object-oriented software. The Singleton pattern is useful when exactly one object is needed to coordinate actions across a system. Singleton patterns allow objects to: Ensure they have only one instance, provide easy access to that instance, and Control their instantiation (for example hiding the constructors of a class).
* **Fulfilling Software Requirements:** The UML diagram supports the software requirements efficiently by:
* Ensuring unique identifiers for games, teams, and players through inheritance and encapsulation.
* Managing the single instance of the game in memory via the Singleton pattern and service layer (GameService).
* Allowing for multiple teams and players within a game through associations.

This domain model provides a clear blueprint for developing the web-based version of “Draw It or Lose It,” addressing the client’s requirements for a scalable, multi-player, and multi-team gaming environment. The use of object-oriented principles aids in creating a maintainable and extensible architecture for the game application.

## [Evaluation](#_2o15spng8stw):

* The below table discusses the potential targets for development. Mac, Linux, Windows, and Mobile Devices are our main host/server options. The table lists the pros/cons, as well as expense, scalability, and tools to go along with each option.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | MacOS are known for there stability and security, making them a good choice for hosting web-based applications however Macs are proprietary and only run on other Apple hardware. They are also more expensive and offer less customization. | Linux is widely regarded for its flexibility, security, and cost-effectiveness in server environments. Linux is considered more secure and comes with many developers’ tools, it is also free. However, it may require more technical knowledge to operate and maintain | Windows is easy to use and is widely used in the business and personal sectors. Windows servers often offer integration with other Microsoft products. It offers a wide base of third-party software. Windows servers are easy to manage and support .NET framework, windows is often targeted by malware | Mobile devices are not typically used for hosting web-based applications due to their limited resources and power compared to dedicated servers. |
| **Client Side** | Mac provides a consistent high quality user experience but can be costly. Development for Mac clients may require specific expertise in Apple’s ecosystem, which would add to the cost of the project. | Linux offers a wide range of distributions that can cater to different client needs. While handling multiple tasks simultaneously without affecting performance. It is cost effective and has a strong support system but may require more time to configure. | Windows has a large user base and supports a wide range of software making it a versatile client-side platform. However, licensing costs and potential security vulnerabilities need to be considered because the licensing fees can be expensive and windows is constantly targeted by malware. | Mobile developments typically involve using Swift for IOS and Kotlin/Java for Android. Cross platform tools like React Native and Flutter are also popular. |
| **Development Tools** | The programming languages primarily used for MacOS are Swift and Objective-C. Xcode is the primary IDE for Mac development . With Mac being proprietary the developers will have to be familiar with Apples ecosystem and guidelines. Xcode is free to use but there is an annual fee for the Apple developer program. | Linux supports a wide range of programming languages including Python, Ruby, and Java. It also supports IDE’s like Eclipse, NetBeans, and Pycharm that are commonly used. Linux’s open-source nature allows for customization, but it requires technical expertise to handle different distributions and configurations. | Commonly used programming languages for Windows development are C#, VB.NET, and C++. Visual Studio is the most popular IDE for Windows development. Windows wide spread use makes it easier to find developers with experience in its languages and tools. Visual Studio has both free and paid versions with the paid version varying in price depending on the level of features required. | The programming languages of mobile devices are Java and Kotlin for Android and Swift, Objective-C for IOS. Other tools include Android Studio and Xcode. Developing for multiple mobile platforms may require more than one team with cross platform development skills. |

## 

## 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**: The chosen architecture should support responsive design, be scalable to handle varying loads, and have robust backend services that can communicate with different client devices. Using Linux servers will reduce the licensing costs and will not limit access to data centers.

Linux offers many benefits such as security features and the operability of the platform. It is also the most common server platform in the industry, it includes many tools, as well as its security software. With the front-end being agnostic it may be written in the preferred language for that platform, such Java for Android, or .NET for Windows.

1. **Operating Systems Architectures**: The chosen architecture should support responsive design, be scalable to handle varying loads, and have robust backend services that can communicate with different client devices and manage the game environment and frontend client-based rendering.

Since Draw it or Lose it is not dependent on twitch reaction it is not necessarily important for there to be low latency between the frontend and backend. Any type of transmission can be done asynchronously. A modern backend containerized microservice like Docker would allow for scalability as needed. The cloud service provider would have to be selected before an exact architecture can be determined due to the providers having some type of proprietary tooling.

When looking at how to reduce some of the costs if we choose to use the front end for rendering this will allow the server to offload some of the more resource intense parts of the application which in turn will help to reduce monthly data center costs. We also need to make sure we are utilizing client-side rendering to help insulate the gameplay from network issues since frame rate is very important for the gameplay. Caching at least the most frequently used images will allow for smooth renderings during the gameplay.

1. **Storage Management:**

HDD’s or SSD’s should be able to provide us the performance needs for the application we are looking for. Again using some type of caching system will help for this part of the gameplay as well. On the server side if we use some type of cloud-native tools, this will allow us more flexibility for scalability and localization.

1. **Memory Management:**

 Linux employs a page cache mechanism, which allows for efficient use of main memory by caching data pages. It utilizes demand paging, a system where only actively used pages are loaded into memory, thus optimizing memory usage. The system decides which pages to replace using the Least Recently Used (LRU) algorithm, ensuring that the least accessed pages are swapped out first.

Android and iOS Memory Handling: Both the Android Runtime (ART) and the Dalvik virtual machine manage memory through paging and memory-mapping (mmapping). This approach ensures that modified memory, such as newly allocated objects or accessed mapped pages, stays in RAM. In contrast, iOS has transitioned from a non-ARC system, where manual memory management was necessary, to an Automatic Reference Counting (ARC) system, where Xcode automates memory management at compile-time.

Server and Client Memory Requirements: On the server side, a baseline amount of RAM is essential for client-side rendering. However, with a containerized microservices architecture, memory costs will scale with user numbers. For client devices, minimal RAM is sufficient as they typically store only a few images in memory at once, alongside the memory required for running the client application, such as a web browser.

1. **Distributed Systems and Networks**: Enhanced Distributed Systems and Network Resilience: The advent of cloud-native architectures is primarily driven by the imperative for high availability and the mitigation of service disruptions. These architectures are designed to be resilient, leveraging the cloud’s distributed nature to replicate and redistribute services across various deployments, thereby safeguarding against extensive downtimes. Furthermore, the decoupling of front-end and back-end components through asynchronous RESTful APIs enhances system robustness. This method of communication ensures that interactions between client and server remain consistent and agnostic of the front-end environment, whether it be Android, Windows, or iOS platforms. It’s worth noting that such architectures also facilitate load balancing and auto-scaling, which are critical for maintaining performance during demand surges and are intrinsic to the cloud’s value proposition.
2. **Security:**

Elevating Security through Strategic Authorization and Encryption: The security framework will be anchored by Role-Based Access Control (RBAC), necessitating the development of a sophisticated entitlements interface to streamline the administration of roles and user accounts. Adhering to the principle of least privilege, the system will confine users to specific game-related functions, such as game initiation, team name generation, and participant registration.

To enhance collaboration, a hierarchical structure may be introduced, delineating roles such as team captains and members, thereby granting certain users the ability to modify team compositions or manage player rosters within their purview. It is imperative to note that no individual will possess ADMIN privileges, ensuring a uniform security posture.

In the realm of API security, robust encryption protocols will be employed, including SHA-256 with 128-bit encryption keys, and any TLS versions preceding 1.2 will be strictly prohibited to maintain integrity and confidentiality of data in transit. Digital certificates, a critical component of the security infrastructure, will be procured from reputable authorities like Entrust. Additionally, the server’s defenses will be fortified with a firewall, configured in accordance with industry-standard best practices, to provide a resilient barrier against unauthorized access and potential threats. This comprehensive approach to security is designed to safeguard the system’s integrity while facilitating a secure and efficient gaming experience.