

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 06/19/2022 | Cody Beck | Updated Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is wanting to expand their game, Draw It or Lose It, to an app with online multiplayer. The Gaming Room is asking Creative Technology Solutions to help with development of the web-based version of their application. The initial design problem is that only one instance of each unique identifier (game, team, player) can exist in memory at any given time. To address this problem Creative Technology Solutions proposes to build the program using singleton patterns for the unique entities, this will ensure that each entity exists only once in memory.

## [Design Constraints](#_2et92p0)

* Allow users to securely connect to the web-based application.

This will require implementation of cyber security measures.

* Application will run on multiple operating systems.

Extra time will be needed to ensure compatibility and to test on each operating system.

* Ensure that only instance of the game, teams, and players exist in memory at any given time.

Using a singleton design pattern will ensure that there are not duplicate games, teams, and players exist.

## [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)

Creative Technology Solutions has prepared an initial UML class diagram. Please see the following image. We use object-oriented programming (OOP) standard practices in all our applications. In this diagram the ProgramDriver class is used to initialize the application and uses the SingletonTester class to test for duplicate instances of a unique GameService class. Following that is the Entity class which is a base class that contains the common attributes id and name. The Game, Team, and Player classes inherit the common attributes from the Entity class and expands to include unique and overloaded methods for each. This design is modular, which allows for easier troubleshooting and changes. Overloading constructors and methods allows us to access them from a common interface for polymorphism. In each class we have private (-) and public (+) attributes. Using private attributes keeps the data from being accessed accidentally thereby encapsulating the data.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac has high-end hardware, secured software, and high-end security implementation. However, the cost to run and maintain a Mac server will be high. If budget isn’t a concern, it is one of the best server environments fast, reliable, and secure. | Linux’s advantage is the cost. It is not as secure as the others since it is open source. It can run security measures that are created by the company, to do this would require an additional team to build and implement it. | Windows hosting is slower than the others. Its security is higher than Linux. It is also the only way to access their proprietary web applications. The cost to host the server falls between Linux and Mac. | Mobile is a great environment for peer-to-peer application development. Hosting, however, would be limited in a client-server system. Many of the features would need to be included in the application, with minimal communication to the server. |
| **Client Side** | While considering Mac clients, having experts in Swift and C++ will be detrimental. Development time will be relatively quick. However, the cost to maintain and update these systems will be higher. | With Linux application development will be faster, but there are some extra processes that need to be addressed. Assembling a team that are versed in C++ and Java will be the first. The second is a team to develop security software. After development the costs to maintain and update the applications will be lower. | Assembling a team of professionals in C++ and Java development will be important. The time needed to develop the application will be average. With the cost of maintaining and updating the application falling in securely in the middle of the spectrum. | The development of the applications for Mobile Devices will require a team experienced in Swift and Java. Development time will be slightly longer which increases the cost. However, maintaining and updating the application will cost significantly less. |
| **Development Tools** | C++ would lead to the best environment to build software on macOS. It is supported by Mac and is an excellent choice for a client-server system. Pairing this with Swift would allow the application to run more efficiently, with fewer translated interactions between IDEs. | In a Linux environment the best IDEs would be C++ and Java. Since Linux supports these languages, integration of the application will be simplified and efficient. C++ is great for building client-server systems. While Java is a good choice to handle the GUI. | An application running in a Windows environment would benefit from the use of Two main IDEs. The first, C++ to handle the client-server system. The second, Java to handle implementation of the GUI. These IDEs are supported and widely used in a Windows environment. | There are two main IDEs used for Mobile Devices. Swift would be used for Mac mobile devices. While Java is the most used for Android mobile devices. These languages are widely used and have support from Mobile Devices Manufacturers making them a great choice. |

## Recommendations

1. **Operating Platform**: We recommend a Microsoft Windows server. This system has sufficient speed to accommodate the “Draw It of Lose It?” game. It will also reduce budget costs and team size. Windows updates its own security measures and processes. It also has a robust file storage and memory architecture.
2. **Operating Systems Architectures**: Windows uses a layered and Microkernel approach to handle tasks. Layered groups similar components in to “layers” and having each layer communicate to the layers directly above and below it. The Microkernel approach operating system components execute outside the kernel with a lower privilege level. This approach exhibits a high degree of modularity. This makes Windows scalable and less prone to failing.
3. **Storage Management**: While using Windows, we have access to their proprietary systems and their FAT file storage system. Having access to FAT will enable fast file storage and access while reducing server storage maintenance.
4. **Memory Management**: Windows uses Contiguous Memory Allocation to ensure that memory necessary for the operating system is always available. This reduces crashing and system latency. Using this method with Dynamic Linking and Dynamic Loading, will keep memory efficiency and speed high. This will ensure a fast and smooth connection for players.
5. **Distributed Systems and Networks**: We suggest utilizing a Client-Server pattern. This pattern uses RESTful API to build distributed hypermedia systems. This network of resources accessible through HTTP that contain addresses of other linked resources. This pattern also uses a text syntax known as JSON. JSON allows us to send and receive information to and from multiple programing languages and Operating Systems. This will ensure proper handshaking between all versions of “Draw It or Lose It?”.
6. **Security**: We suggest Dropwizrd BasicAuth security and the Dropwizard Jersey/HTTP configuration libraries to secure the server. Implementing these will ensure each level of access is passworded to prevent unauthorized users from accessing data above their access level.