

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)4

[**Recommendations**](#_m8aleynsvzvc)7

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 02/20/2022 | Jessica Ayer | Updated recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is transforming their Android game app, *Win, Lose or Draw* to a web-browser game which needs to function on all OS web browsers. Game instances themselves cannot have duplicate copies existing in memory at the same time therefore, must have a unique name and id that can be cross checked with already existing names and ids. The software must be developed in a way that it can check those instances of games identifiers to prevent duplicates. Each game must only allow unique teams and may have multiple unique teams which may have multiple players. Potential team names and ids must be cross checked with already existing names and ids to ensure that no duplicates exist. To accomplish this, pre-existing identifiers for both game instances and teams will exist in separate lists that the program can iterate through to check for duplicate entries. User’s will be asked to provide unique game identifiers and a new game instance or team will only be created if there is no pre-existing name or id that matches the user’s entry.

## [Design Constraints](#_2et92p0)

* Must prevent duplicate game instances from being created
* Must prevent duplicate team name entries
* Games must be able to have one team or multiple teams
* Teams must be able to be multiplayer
* Must work on all OS platforms web browsers (previously only available as an Android app)
  + Graphics adjusted to larger screens
  + Touch buttons redesign for use with mouse
* Using JAVA for base language

## [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 package will all be held in a single package but will import a few of JAVA’s built-in methods including array lists and the iterator function. The parent class Entity will be created and common the attributes amongst the Game, Team, and Player classes will be moved into the new parent class. Those attributes will include Id and name which will be encapsulated or private. The Game, Team, and Player class will be related through association and will now inherit the two attributes to ensure that, via use of the Singleton design, no two game instances and teams are the same. An infinite number of games, teams and players may exist within an Entity and an infinite amount of game services may exist withing a game however, none can have duplicate ids or names.

The game class will hold a list of teams, be able to search through pre-existing team names for matches to the user input and have the ability to add new teams should the user provide a unique identifier. It will get the user’s name and id from the parent class, Entity. The Team class will hold a list of players, be able to search through the list of players for pre-existing player names for matches to the user input, and be able to add new players should the user provide a unique identifier. The Game, Team, and player class will all utilize polymorphism on the toString method so that dependent on which class method is called, different output strings will be produced. The GameService class will hold a list of active games and identifiers for getting the next game id, team id, and player id. It will create new instances of game services, be able to add games, game ids, game names, game counts, player ids and team ids. We will then use a ProgramDriver class to hold our main method and a SingletonTester class to test the programs functionality.

**"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** | Pros:  - Few security issues with malware. Built in firewall, “Gatekeeper”, and XProtect.  - Unix based: stable platform.  - Seamless integration between hardware and OS.  -Supports ADP cloud services.  Cons:  -Does not offer Web hosting, must use third party host.  - Supports LDAP but user must manually configure.  -Limited RAM and no dedicated graphics processor/upgrading hardware is limited. | Pros:  -Linux web server hosting works on all OS.  -Fewer security risks. Uses access Controls.  -Unix based: stable platform, can run for several days without rebooting.    Cons:  -Known server configurations issues.  -Unreliable version support timeframe.  -Third party cloud services must be used. | Pros:  -offers server-based deployment, WDS  -Windows web server hosting works on all OS.  - Easily upgradable hardware.  - Uses security tokens.  -Supports ADP and LDAP cloud services  Cons:  -More Malware security threats then other OS.  -Known for lack of stability.  -Every program requires its own update mechanism. Due to use of different OEMs more susceptible to compatibility and optimization issues. | Pros:  -Third party applications can transform mobile devices into web hosts.  -Use of passcodes/encryption.  -Supports ADP and LADP cloud services.  Cons:  -High security risk, used on insecure networks and devices are frequently lost, User is responsible to setting most configurations.  -Upgrading typically entails purchasing a new device. |
| **Client Side** | Pros:  - Simple GUI  - Supports Cross-Platform gaming (currently 71 games exist)  -Simple platform to learn.  Cons:  -Cross platform deployment is expensive.  -MacOS is not designed for gaming. | Pros:  -Free/open source. There is no license cost.  -Supports Cross Platform gaming (however only 31 games currently exist)  - Easy to configure.  Cons:  -Difficult to use for those lacking technical experience. | Pros:  -supports Cross-Platform gaming.  -Established OS for gamers/larger gamer market.  -Simple GUI  -Currently most used OS and easy to learn.  Cons:  -Mid-range cost for cross-platform development. | Pros:  -Lower price options for cross-platform development (however prices may vary drastically depending on app needs)  Cons:  -Applications must be simplified.  -UI/UX requirements must be more flexible and redesigned for small touchscreen input. |
| **Development Tools** | Server-side Languages:  JavaScript, PHP, Python  Client-Side languages:  HTML, CSS, JavaScript, Python, PHP  IDEs:  XCode, CLion, NetBeans, PHPStorm, IntelliJ, Visual Studios, PyCharm, Eclipse, Komodo, Cordova | Server-side Languages:  JavaScript, SQL, PHP, Python, Ruby  Client-Side languages:  HTML, CSS, JavaScript, Python, Ruby on the Rails, PHP  IDEs:  CLion, Netbeans, IntelliJ, Visual Studios, PyCharm, Eclipse, Komodo, Cordova | Server-side languages:  JavaScript, SQL, PHP, Python, C++  Client-Side languages:  HTML, CSS, JavaScript, Python, C++, PHP  IDEs:  CLion, Netbeans, IntelliJ,  Visual studios, PyCharm, Eclipse,  Komodo, Cordova, | Server-side Languages:  JavaScript, PHP, Python, C++  Client-side languages:  HTML, CSS, JavaScript, Python, C++, PHP  IDEs:  CLion, Netbeans, PhpStorm, IntelliJ, Visual Studios, Eclipse,  Android Studio, Komodo, Cordova |

## Recommendations

1. **Operating Platform**: My recommendation would be to use Microsoft Windows as the primary platform for “Draw It or Lose It”. Windows would be ideal for the client because it is designed with game play in mind. Teamed with their Azure cloud service, Windows can support a powerful multi-tiered web application. It is also the most used operating system in U.S. homes and it supports cross-platform gaming which will make transferring the game to other platforms less challenging. Windows has the largest network of game players including the Xbox community which gives the client a higher level of marketability. This also makes Window the ideal platform for the users. Window’s though not the cheapest option, is middle of our three and offers the most cost/benefit for the client. Using their cloud service Azure offers the option to scale services accordingly as the game grows improving performance while reducing excessive cost to the client.
2. **Operating Systems Architectures**: Microsoft windows is a single processor system that utilizes a layered design. It consists of a Kernel mode and user mode. The Kernel contains the hardware abstraction layer, the drivers, and microkernel. The user mode cannot directly access the hardware or memory creating a layer of security. The user mode includes an environment subsystem that makes cross-platform software possible. It also has an integral subsystem which handles security and workstation services. If a multi-tiered system is implemented, then having a web tier to host the user interface will improve performance and security by using a cloud base server for frontend services and a windows server for backend services.
3. **Storage Management**: I would recommend storing the backend code on a Windows Server and frontend storage on a cloud server. This offers our client a higher level of security without requiring them to maintain massive in-house servers that would require expensive upkeep. Azure has several storage options to fit various needs. Their Premium SSD offers backup recovery creating incremental snapshots and redundancy options which allow the client to store copies of data across Azure’s 54 server locations. Microsoft offers Synchronous Multiplayer solutions via Azure Kubernetes Service which has a solid reputation for game development. Kubernetes has a built-in StorageClass that utilizes a storage cluster and pod system where the admin defines the storage cluster and the user places requests to access individual pods. These clusters can be accessed cross platform with a public IP.
4. **Memory Management**: When working with a cloud server, it will be vital to use a system memory usage ratio metric to monitor memory usage and prevent a system crash. An “Eviction policy” is recommended which can be implemented by removing less recently used data. An expiration can be set on the data that is not needed to be proactive in clearing out the cache. Eviction can lead to fragmentation so utilizing the provided activedefrag configuration will help prevent memory issues. Azure utilizes caching which allows for temporary storage on the user’s device bringing needed data into main memory. This helps with speed and performance when using a cloud server. It is important to make sure that the cache is cleared appropriately to minimize the impact on the user’s device. In the game “Draw It or Lose It” completed images would no longer need to be stored and could be evicted from the memory cache.
5. **Distributed Systems and Networks**: On an individual player level, Azure PlayFab utilizes various forms of authentication which allows players to be using multiple types of devices a user can play “Draw It or Lose it” on their computer and then switch to their phone without losing their personal stats. Broaden to a multi-user system, Azure Kubernetes can be connected to Azure Arc which uses a “multi-cloud blockchain distributed ledger technology”. This allows each party to use separate cloud providers and still have a fluid experience. Azure’s networking services uses branch connectivity through Virtual WAN which allows remote users to be using android, iOS, Windows, or Linux systems.
6. **Security**: Microsoft Azure Kubernetes API sever components utilizes REST APIs and Webhooks which validate requests to the server. Access codes are used to protect the API methods and stored in “Kubernetes Secret” apiaccesscode. Azure also offers additional features including a team of cyber security experts to help protect the users as well as what they refer to as disaster recovery. Some of the addition security features that are provided include Microsoft Defender for Cloud, Azure Active Directory (identity management), Azure firewall, Azure Key Vault (secure token, password, certificate, and API key storage), and Azure application Gateway (traffic management).