

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 |
| --- | --- | --- | --- |
| 1.0  3.0 | 03/20/22  04/17/22 | Devyn Mustard  Devyn Mustard | Initial Release  Added Recommendations |

**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 currently has an app called Draw It or Lose It, this app is currently only found on Android devices. They would like to expand this app and make it web based. Making the game web-based will allow it to function on multiple platforms, seamlessly. The app hosts different game instances, each with their own unique sets of teams and players. The staff at The Gaming Room do not know how to start setting up this cross-platform environment.

## [Design Constraints](#_2et92p0)

Each platform has different technological requirements, and limitations.

Making an app cross-platform requires building with separate API’s or SDK’s, to coincide with requirements.

Game needs to allow users from different platforms to connect seamlessly.

Game needs to verify that each instance, team name, and player name are unique.

Utilize a set of indexed Id’s to confirm each set is unique.

As the game is timed, it must be able to operate across multiple platforms quickly, with minuscular latency.

## [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 ProgramDriver class contains the main method and drives the program. ProgramDriver is connected to SingletonTester to make sure there is only one instance of GameService. SingletonTester is responsible for making sure only one unique instance of GameService is running. The Entity class has a selection of children classes, the Game class, Team class, and Player class. These three children classes extend/inherit from Entity, and its attributes. A game can have a number of team objects, and a team can have a number of player objects. The GameService class hosts the unique instance of a game, due to the class utilizing the singleton design pattern. To make sure each instance of a game, each team name, and each player name is unique, we utilize the iterator pattern. This not only verifies that each instance is unique, it also assigns a unique Id.

"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** | Macs do have server hosting capabilities but fall short of both Windows and Linux due to price and setup. There are also licensing fees for developing with a Mac. | Linux has extensive hosting capabilities, being cheap, reliable, and relatively easy to set up. Linux can also be very lightweight, as well as the cheapest option. | Windows has server hosting capabilities, that are both quick and easy to setup. While not as cheap as Linux options, it offers easier setup. | Mobile Devices are not equipped to host a project of this scale, as they would not be reliable. Resulting in outages and downtime. |
| **Client Side** | To remain compatible with Mac users, you must make the website compatible with the Safari web browser. This means making sure all HTML/CSS elements can be displayed on the Safari browser. The app would need to be responsive to fit on the screens of laptops. | To remain compatible with Linux users, the front-end website must be compatible with the main Linux web browsers, Google Chrome, Vivaldi, and Firefox. | To make the web app compatible with Windows, make sure the web App supports all main browsers, like Chrome, Edge, and Firefox. The web app has to be responsive for smaller screens, like Laptops. | Remaining consistent with the other OS, it just needs to support the main web browsers. The website will need media queries to adapt and remain responsive to vastly different screen sizes and resolutions. |
| **Development Tools** | XCode is the IDE of choice for Mac development. Utilizing the programming language Swift to create the backend of the web app would be an option. Yet this option can be costly with certain licensing options. | Eclipse IDE could be used for a Linux environment. Utilizing Java to create the backend. This option would be free and would only require one team. | Visual Studio would be the IDE of choice on Windows. Utilizing a language like C# with the .NET framework to drive the backend. This would require only a single specific team for backend development. | No specific development needs to be done for the backend of the app for mobile, as mobile devices will only interact with the front end. Making the front end responsive to mobile devices would require a team of web developers. |

## Recommendations

1. **Operating Platform**: I recommend using the Windows operating system to host Draw It or Lose It. While it is more expensive than Linux, it is not as costly as hosting via MacOS. Windows is more compatible with being cross-platform, as you can deploy applications that can be run on a high number of different platforms. Using Java and an Apache server with Tomcat, you can host, develop, and set up a web application all within the Windows OS. Keeping costs down to not have to facilitate multiple different platforms to host and develop, with Windows it can all be done with one.
2. **Operating Systems Architectures**: Windows is not as lightweight as some Linux builds, as the OS does take more system resources than smaller Linux builds would. Yet Windows has a comprehensive firewall that will keep networking ports closed and secured unless explicitly opened. Windows is also fully capable of multithreading, allowing for more processes to be run simultaneously, allowing for the web application to take and process more user requests. Allowing for faster data manipulation, data management, and for calculations to be completed.
3. **Storage Management**: Storage for Draw It or Lose It can be done on the local machine that is hosting the web application. Done by utilizing the system’s drives to store player data, this way the data can easily and quickly be pulled by the web application, without needing to host a separate database that the web app then needs to pull from. Allowing for faster read and write times, and allowing all information to be stored on one device, cutting down on potential costs.
4. **Memory Management**: Memory will be managed by the local machine that is hosting the web application. Utilizing the system’s RAM will allow for the web application to instantiate variables, manipulate data, and take data sent from connected users. Java inherently utilizes automatic “garbage collection” that will automatically deallocate memory that is no longer needed. This will keep the web application from utilizing too much of the system’s resources, keeping its’ memory effectively managed automatically.
5. **Distributed Systems and Networks**: Using an Apache Tomcat server, the web application can be connected to by any platform that is able to connect to a website via a web browser. Making sure the game is responsive would be the only step so that each system can feel like a native experience, as it is a web application, it provides seamless connectivity from any platform. The host machine does need enough processing power to be able to efficiently handle every request that is sent to it, especially with multiple games being hosted at a time. Handling requests efficiently through directed traffic and multi-threading will allow the system to keep up with each player request as long as the machine has a proper amount of resources, else it will be throttled. Outages are always a possibility, so having the system hosting off a secure and stable network will help minimize outages, as well as having the host machine and router on an uninterruptible power supply (UPS) will minimize hosting outages due to power surges or outages.
6. **Security**: Hosting on a secure network, with only certain network ports open to direct user requests will keep the web application secure. Writing code with security in mind will keep user data safe, making sure encryption is used for anything being sent over a network. This way only the host machine can decrypt user data that is sent, so nothing can be intercepted and decrypted. Having the ability to connect to the web application from any device, any input needs to be encrypted, and stored where only select users have access to. Keeping connected user’s IP hidden as well as sensitive information from other users will also keep user’s data safe, only display what is needed and nothing more. This can be done and displayed by obtaining an SSL certificate, as it allows for an encrypted connection for all users connected to the web app, keeping user data and information secure.