

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.3

## 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)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/07/22 | Cooper Adams | Updated Executive Summary and Design Constraints |
| 1.1 | 11/11/22 | Cooper Adams | Updated Domain Model and Table of Contents |
| 1.2 | 11/26/22 | Cooper Adams | Defined the evaluation of the different platforms for hosting the game app |
| 1.3 | 12/7/22 | Cooper Adams | Defined the recommended choices and processes for the core structures of the game and server |

## [Executive Summary](#_sbfa50wo7nsh)

The staff of The Gaming Room are unsure how to facilitate Draw It or Lose It in a web-based environment. We will be doing the beginning development with Java, and will ensure that the game will only have one instance in memory at a time with unique identifiers and names for games, teams, and players.

## [Design Constraints](#_2et92p0)

In a web-based environment, the data for the game (pictures, names, IDs) will not be stored locally on the user’s machine, so the game can only have one instance in memory so that it is not cross-referencing data through multiple instances at the same time.

The game will also need to function with mouse and keyboard rather than the touchscreen capabilities of the Android app.

The Gaming Room already has an existing database of images to be rendered which is used in the mobile app, so the web-app will also have to use it.

## [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 “heart” of the model is the GameService class. It contains the list of games and most of the relevant game information. Its variables are private and static, and its’ constructor is private, making it a singleton class. Due to GameService being a singleton class, only one instance of the game service can exist in memory at any time (Software Requirement Four). Game, Team, and Player are all extensions of the Entity class, which demonstrates the OOP of inheritance. This is done to reduce redundancy throughout the code, as Game, Team, and Player all carry an ID and a Name. Relationally, the GameService can have many Games, a Game can have many Teams (Software Requirement One), and Team can have many players (Software Requirement Two). This is demonstrated not only by the multiplicity on UML diagram, but by the Lists in GameService, Game, and Team, which each hold a list of unique players, teams, and games (Software Requirement Three). Finally, there is the ProgramDriver and the SingletonTester, which are for testing purposes. The ProgramDriver uses the SingletonTester to prove that there is only one instance of the game service active.

**"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

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is a solid choice for hosting a server on the development side as it is easy to configure and make changes to. The major downside is that it is available on a comparatively small set of hardware, and MacOS may not support some pieces of software that could help in developing or maintaining the server. | Linux is the most popular choice for hosting due to its characteristics. Linux as a platform is very secure by default. It would likely be more cost effective as well. What could be a potential weakness, is that working with the command line and manually editing config files typically requires experience on the platform. | Windows is generally considered the second choice for web hosting, likely due to familiarity with the platform. It is much less secure than Linux but would have compatibility with most software. Windows is also not free, so it would cost more to host than using Linux. Windows Server would be the choice for hosting, and the Standard package is $577.99 while the Datacenter package is $3,289.99. | The mobile app will likely have to use a server setup on a different platform, as mobile servers are not commonplace. You can use something such as Jetty for an HTTP server, but it will likely be limited in deployment across the board and would also require the mobile device hosting it to be always on, which isn’t feasible given the comparative lack of power, and it |
| **Client Side** | Machines that run MacOS will be the most expensive, and the team would require a few to test on, so Mac development would be the highest cost. Expertise would be needed in navigating MacOS and likely Safari, as it is only on the Mac platform. The timeframe for development would likely be a few months as the team would work to unify the front-end’s visuals and functionality on each platform. | Linux is of no cost to install, and many developers likely already use it in their day-to-day work. Most Linux software is open source, so if it is needed it should not cost extra. Expertise in navigating and developing for Linux is required and is likely more intense than Mac or Windows. Once again, the timeframe for development would likely be a few months as the team would work to unify the front-end’s visuals and functionality on each platform. | Windows is a great platform for web-app development as it is easy to navigate and is generally very compatible. Windows devices, such as a Dell or HP desktop, would be cheaper than a Mac system too. Compared to Mac and Linux, minimal expertise with the system would be required due to familiarity with Windows. Once again, the timeframe for development would likely be a few months as the team would work to unify the front-end’s visuals and functionality on each platform. | The mobile app versions would likely take more time to finish and would generate additional costs. Expertise is required in IOS and Android development which differ vastly due to the languages of choice for these devices. To publish on the Google Play Store, a one-time fee of $25 is required, and Apple charges a yearly $99 for their development license. The mobile app would likely take longer than the web-app to finish development as it would need to be modified for the smaller UI of phones and tablets and could potentially be rejected when submitted for publishing on the app stores. |
| **Development Tools** | Most Mac development is done in Objective-C but developing a web-app requires HTML (plus CSS and potentially JavaScript). The built-in command line is extremely useful, and Mac has access to its own version of VS Code, which is completely free and has many great extensions that are helpful for development, which makes it the most relevant for developing the front-end of the web-app. Safari is an extra browser that must be tested on Mac devices, so it is also needed during development. | Like Mac, Linux developers will spend a lot of time on the command line, and they also have access to VS Code for Debian, Ubuntu, and more. Linux has access to most major web browsers now, including Microsoft Edge about a year ago, so each can be used for testing. | Windows is the holy grail of software compatibility, and has access to most any IDE, including VS Code or Visual Studio itself. VS Code is still the preferred choice as it is lightweight, free, and carries great extensions for web development. Windows has no shortage of mainstream browsers to test the web-app in either. | IOS app development is typically done in Swift in the Xcode IDE, which is free. Android apps are made using the SDK framework, and can be written in many languages like Java, Kotlin, C, C++, etc. The IDE is left up to the developer, but Visual Studio is a common choice due to its tools and emulation capabilities, and it is also free. Mobile app development is not as simple as scaling down a web page, so there will likely need to be three teams: one for the web-app, one for IOS development, and one for Android development. |

## Recommendations

1. **Operating Platform**: The recommended server platform would be Linux. Once setup properly, the server should be easy to maintain and will be able to communicate with the web-app on other computing environments.
2. **Operating Systems Architectures**: Linux is secure by default, free to install, and open source. Many popular pieces of software are now on Linux, so the development team should not be limited in that facet if needed. Most of the navigation and setup work would be done in the terminal, which is a command line interface. Linux also carries authentication features, which will protect the data of the developers and users alike.
3. **Storage Management**: A database (SQL or noSQL) should be used to keep track of user data, including but not limited to usernames, passwords, emails, and game statistics. The images to be rendered during gameplay should be kept in the server’s file system, though cloud storage is also strongly recommended as mass image hosting is inexpensive.
4. **Memory Management**: The server itself should have access to at least four gigabytes of RAM, preferably eight for guaranteed performance. The incoming requests to the server should not be heavy, but the server would also be working in conjunction with a database like MySQL, which does take a bit more memory to run smoothly. To lessen the load on the server and maintain the game’s performance, the apps should make use of Client-Side Rendering and caching. Client-Side Rendering would have the browser or app do all of the graphical rendering for the game, which would save the server a bit as there would be many requests from each instance every time an item would need to be displayed. Caching helps the game’s performance as it would have the user’s system “remember” the items that would be loaded frequently, like logos, icons, banners, and even colors so that they do not have to be re-rendered every time they are needed.
5. **Distributed Systems and Networks**: First, the server must be running for there to be communication across platforms. Additionally, the server and the players must have internet access in order to communicate. The server is dependent on the database, which is local to the server. It communicates with the database through local accessor functions when a request is made from the client side to store, retrieve, and verify user data. Across each platform, the game will be using HTTP requests to access the server. No extra software or components should be needed for the game to function, but the developer team should include specific error messages for potential problems (no internet, server down, user data not found, etc.).
6. **Security**: Linux is a naturally protected platform, so the server has some assurance there, but it is best to take other security measures. All data sent back and forth between the client and server should be encrypted so that sensitive data cannot be stolen by watching the data stream. In addition, the team should employ measures to prevent SQL injections, which are very common and considered to be an “easy” way to steal data. All input should be put into prepared statements with parameterized queries so that hackers cannot “change” the SQL request’s meaning. In addition, each request should be put through strict input validation to ensure the proper data is being accessed based on the type of request being made. With all of this in mind, the development team should have no problem making the game, server, and database secure from threats.