

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.3

## 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 |
| --- | --- | --- | --- |
| 1.3 | 2023/12/10 | Dalton Rose | Finalized design document. |
| 1.2 | 2023/12/07 | Dalton Rose | Initial draft of the Recommendations section of the design document. |
| 1.1 | 2023/11/21 | Dalton Rose | Initial draft of the Evaluation section of the design document. |
| 1.0 | 2023/11/13 | Dalton Rose | Initial draft of the Executive Summary, Requirements, Design Constraints, and Domain Model sections of the design document. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room, developers of the game Draw It or Lose It, want to develop a new web-based version of the game that supports multiple platforms.

## Requirements

* The game should be playable on multiple platforms.
* Each game can have one or more teams, each consisting of one or more players.
* Game and team names must be checked against existing game and team names to ensure they are unique.
* Only one instance of the game can exist in memory at a given time.

## [Design Constraints](#_2et92p0)

* Client/Server Architecture – For a web-based multiplayer game, we’ll need to utilize develop both client and server software.
* Programming Language – For this project, we’ll need to use a programming language that can support web applications like a game.
* Platform Variety – As we’re targeting multiple platforms, we should take into consideration the differences between various web browsers and screen sizes.

## [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 Entity class which has private attributes “id” and “name” and multiple private and public functions associated with it. The Entity() function is a constructor, getId() and getName() are accessors which return the id and name of the Entity, and the toString() function returns information about the Entity as a string. Three classes inherit directly from the Entity class: the Game class, the Team class, and the Player class.
* In addition to the attributes and functions inherited from the Entity class, the Game class has a private attribute, “teams”, which is a list of teams associated with the game. It additionally has several public functions including a constructor function Game(), the addTeam() function to add teams to the game, and the toString() function which converts returns information about the game as a string. The Game class is associated with the Team class in that a number of Teams are involved in a Game.
* In addition to the attributes and functions inherited from the Entity class, the Team class has a private attribute and three public functions. The attribute is “players”, a list of players on the team. The Team() function is a construct, addPlayer() adds a player to the “players” list, and toString() returns information about the team as a string. The Team class is associated with the Player class in that a number of players are on a Team.
* In addition to the attributes and functions inherited from the Entity class, the Player class has two public functions. Player() is a constructor function whereas toString() returns information about the Player as a string.
* The GameService class has many attributes and functions which are used to store and manage information about all Game objects. Only one instance of the GameService class can exist at a given time.
* The principles of object-oriented programming are utilized in this application. Encapsulation is used in this program by keeping attributes private and by using mutator or accessor functions to modify them. This is important because it prevents attributes being modified unintentionally. Inheritance is used by the Game, Team, and Player classes all inheriting common attributes and functions from the Entity class. This is important because it reduces the amount of work needed by eliminating unnecessary repetition. Polymorphism is used to have functions which perform different tasks based on the argument. For example, the getGame() function of the GameService class behaves differently whether it is provided with an id (long) or name (string). Abstraction is utilized in how the Classes and their functions are broken up. For example, while a game contains many teams, the code relating to teams is kept in the Teams class and only the code directly relevant to games is kept in the Game class. This is important because it helps ensure the code is easy to manage by abstracting away anything we aren’t working on at this moment.

**"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)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | As of 2022, Apple discontinued macOS Server. While existing software may continue to work for some time, it would be unwise to use a server platform that has no official support moving forward. For this reason, macOS has only disadvantages in hosting a web-based software application. | Linux is widely used to host servers for a variety of software applications. Unlike Windows, Linux is free and has no licensing fees. Linux allows for more customization and has a wide variety of software support. Additionally, Linux is less resource intensive than the alternatives. The only disadvantage is that Linux may be less intuitive to inexperienced users. | Windows Server has much of the same capabilities as Linux. The one advantage over Linux is that the GUI may be more intuitive to users who have more experience with Windows than Linux. However, the drawback is that Windows Server is more resource intensive. Additionally, there is a licensing fee. | While mobile devices are sometimes used to host servers by hobbyists, they are not commonly used to host professional software applications. When hosting software applications, reliability is important so using a device reliant on mobile service and a battery is not advisable. |
| **Client Side** | As we are targeting a web-based software application, the application would run in the web browser. This means that it would be relatively simple to support Mac as a platform as it should already work in common web browsers like Chrome, Firefox, and Edge. Some additional time should be spent ensuring the application runs properly in Safari as that web browser is more commonly used on Mac. | As the application runs in the browser and should adhere to proper web standards, supporting Linux as a client is relatively simple. Much of the work being done to ensure the application works correctly in Chrome, Firefox, and Edge is transferrable to the user experience on Linux. Supporting Linux as a platform would require more time spent testing functionality but should otherwise not be a major hurdle. | Much of the development work is portable in that if the web application is developed properly, not much development time should be dedicated to any individual platform. Browsers function similarly across operating systems. Time should be allocated for Windows testing but much of the development should be cross-platform. | While browsers do function similarly across operating systems, mobile devices like smartphones are a slight exception. Considerations must be made to ensure the page displays properly on devices with different screen sizes and that the web application functions properly on mobile devices like tablets and smartphones. |
| **Development Tools** | As we’re developing a web-based application, we can continue to use Java and Eclipse. However, if we were developing a native Mac application, we’d likely want to use the Swift programming language and either Visual Studio Code, CodeRunner, or Xcode as our IDE. | Our combination of Java and the Eclipse IDE is still relevant for our web-based application. However, if we were developing a native Linux application, we could continue to use Java or use another programming language like Python or C++. In this case, we could use IDE’s like PyCharm for Python or Visual Studio Code for either. | For our use, Java and Eclipse are still relevant when developing a web-based application like Draw It or Lose It. There are a variety of other programming languages and tools we could utilize when targeting Windows as a platform. For example, C++ and Visual Studio Code, Python and PyCharm, and JavaScript and WebStorm. | For native iOS development, Swift paired with a compatible IDE like Visual Studio Code, CodeRunner, or Xcode is recommended.  For native Android development, Java and Kotlin are the most common languages for development. We might utilize an IDE like Android Studio, Visual Studio Code or Eclipse. |

## 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**: Draw It or Lose It will utilize the Linux operating system to run the game servers. Linux is ideal for this project due to lower costs and higher performance than alternative operating systems. Draw It or Lose It should utilize cloud infrastructure for the servers to ensure the game is scalable.
2. **Operating Systems Architectures**: Draw It or Lose It will utilize client-server architecture to enable the game to support multiple platforms. The Draw It or Lose It client will run on multiple platforms across PC (Windows, Linux, Mac) and mobile (iOS, Android). The Draw It or Lose It client will communicate with the server software in order to function.
3. **Storage Management**: Draw It or Lose It will utilize both local and cloud storage. Essential data will be stored locally while certain assets will be stored in the cloud and only downloaded when needed. To reduce unnecessary bandwidth usage, assets should be cached so they only need to be downloaded once. By using cloud storage for assets like images, we can keep the initial download size relatively small which helps to make the game more accessible to users on mobile devices with limited storage.
4. **Memory Management**: Draw It or Lose It should only load data into memory when it is necessary for the game to function. For example, the game should not load every possible image into memory but only those currently being used. This ensures that Draw It or Lose It can be supported on more devices because older mobile devices are likely to have limited memory.
5. **Distributed Systems and Networks**: By utilizing cloud infrastructure to host the game servers, we can ensure that Draw It or Lose It can scale to meet the needs of the players. We can have a server dedicated to the lobby service and spin up additional servers to host more games as needed. An additional benefit of hosting our servers using cloud infrastructure is that there are less worries regarding uptime and security.
6. **Security**: User authentication for Draw It or Lose It should be handled through a username and password system with optional two-factor authentication. These passwords should additionally be salted and hashed to ensure that user information is secure even if our database is somehow breached. HTTPS should be utilized to ensure communication between the client and servers is secure. There should also be checks to ensure requests are coming from the correct client and that the server only responds to valid requests. Regular backups should be made in case of data loss. We can utilize the security features built into Linux such as the permissions system to minimize any risk of unauthorized access. It’s also important that the servers are kept up to date and are configured correctly. Using cloud infrastructure will also provide us with some security benefits and also protection against DDOS attacks.