

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.gjdgxs)1

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.3znysh7)2

[**Executive Summary**](#_heading=h.2et92p0)3

[**Design Constraints**](#_heading=h.tyjcwt)3

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

[**Evaluation**](#_heading=h.2s8eyo1)3

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/16/2022 | Johnathan Wood | Initial document creation |
| 2.0 | 10/2/2022 | Johnathan Wood | Project two additions |
| 3.0 | 10/15/2022 | Johnathan Wood | Project three additions |

**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](#_heading=h.2et92p0)

Draw It or Lose It is a web-based game requiring players to take turns guessing puzzles based on presented images to gain points. Players can play individually, or on teams, and a game consists of four 1-minute rounds. If a team is unable to guess their picture in the allotted time, the other teams have 15 seconds to guess the puzzle and “steal” the point.

CTS has devised a solution to implement Draw It or Lose It using a Java backend to handle game and player creation and management. This software solution can be implemented on a variety of hardware and operating systems, for which options will be explored later in this document.

## [Design Constraints](#_heading=h.tyjcwt)

* A game will have the ability to have one or more teams involved.
  + A Game object cannot be instantiated without at least one team.
* Each team will have multiple players assigned to it.
  + A Team object cannot be instantiated with less than two players.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
  + The GameService class manages all game and player instances and ensures that any new game and player instances have unique names.
* Only one instance of the game can exist in memory at any given time.
  + This is accomplished by creating unique identifiers for each instance of a game, team, or player, and using a singleton pattern for the game manager so that all game instances are available in the same location.

## [System Architecture View](#_heading=h.3dy6vkm)

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](#_heading=h.1t3h5sf)

The UML class diagram provided below explains how the classes relate to each other and identifies the object-oriented programming principles used to fulfill the aforementioned software requirements. The com.gamingroom package contains several classes that work in tandem to achieve the goals of the application. The GameService keeps a list of Game objects, and provides methods to create games. Game objects inherit properties and methods from an Entity base-class that encapsulates properties and functions common to Games, Temas, and Players. The Game object also keeps a list of Teams, and provides a function for adding a new Team. Team objects, aside from the inherited Entity attributes, keeps a list of Players on the Team, and provides functions for adding new Players to the Team. Finally, the Player class is simply a wrapper of the Entity class, used to provide better readability in the code.

**"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](#_heading=h.2s8eyo1)

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** | Mac OS is proprietary, and runs only on costly specialized hardware. While it does support Java and web hosting, this platform is not advised for use in this application. | Available in various flavors, Linux is the most common server operating system in the world. It is capable, highly configurable, and in many cases, available for free, making it ideal for applications such as this. | Windows offers server-flavored versions of its operating system, and may be easier to learn for those unfamiliar with linux. It would be a good second choice for this application. | Mobile devices, with their limited resources and ever-changing network conditions would not be ideal for hosting websites or server-side applications. |
| **Client Side** | Mac OS is compatible with all major web browsers, and since the application is web-based it can be used from any device with a compatible browser. | Linux is compatible with all major web browsers, and since the application is web-based it can be used from any device with a compatible browser. | Windows is compatible with all major web browsers, and since the application is web-based it can be used from any device with a compatible browser. | Mobile platforms such as IOS and Android offer many major web browsers, and since the application is web-based it can be used from any device with a compatible browser. Platform-specific apps may also be created to provide a more “native” experience, and integrating them with the REST API on the server-side is straightforward. |
| **Development Tools** | Aside from the higher upfront cost and generally limited software compatibility, Mac OS is fairly well-suited for application development using tools such as Eclipse or IntelliJ for Java, and VSCode for client-side development. | Nearly any device can run Linux, it’s free, lightweight, highly configurable, and is very well-suited to application development using popular IDEs such as Eclipse, IntelliJ, and VSCode. | Windows is the most common “desktop” OS in the world, and is therefore familiar to many people. This, combined with great software availability make it well-suited to application development using popular IDEs such as Eclipse, IntelliJ, and VSCode. | Mobile devices, with their small screens, relatively short battery-life, limited resources, and ever-changing network conditions would not be ideal for software development. |

## Cost Considerations

* Implementing responsive web design concepts on the client-side combined with the utilization of modern CSS, libraries, and web frameworks allows one web application to deliver a seamless, cohesive experience to desktop and mobile users alike. This approach also eliminates the time-cost, complexity, and the often less-than-impressive user experience of redirecting mobile users to a second “mobile-friendly” version of the site.
* Thanks to the use of the client-server architecture and REST API for the server-side Draw It or Lose It will also be able to support cross-platform play, allowing users on various types of client devices to connect to join games with players on other platforms, since all of the clients connect to the same backend, further reducing development costs.
* However, development of Draw It or Lose it will likely require multiple software development teams, as client-side, server-side, and mobile application development are all very specialized disciplines and it is unlikely that a single team could be assembled with all of the expertise required for all three. Furthermore, Android and IOS development are also very different, and may require separate teams. These various teams will need to work very closely to ensure a cohesive cross-platform experience is attained.
* There are also costs associated with the tools and platforms chosen for server OS and some development tools. MacOS, Windows, and some enterprise flavors of Linux require a license for use, along with some development tools such as IntelliJ, though there are free alternatives.

## 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**: CTS recommends Linux for hosting the server-side application, and for development of the software. As mentioned previously, its free-to-use philosophy coupled with ease of configuration and maintenance make it ideal for applications such as this. Additionally, CTS believes that AWS EC2 is an ideal candidate when compared to the upfront cost, setup and maintenance of server hardware. Using EC2 ensures a stable, performant, secure platform that can handle both the server-side application and the hosting of the web-based version of Draw It or Lose It.
2. **Operating Systems Architectures**: By employing EC2 as the hosting platform, we are able to abstract the hardware away and really focus on the OS and software level. The architecture is simple, an EC2 instance configured to run as a web server and host the backend code is prepared and deployed in each region the game wants to serve. EC2 even allows for scaling the instances per-region so that they can be cost-optimized independently. Players use the app, or go to the website, log in, and search for or start a game, play, and repeat!
3. **Storage Management**: Starting with storage management on the server-side, there is the challenge presented by the images that must be handled effectively. 200 Images at 8Mb each is nearly 2Gb of data, which isn’t a ton of space, but when you think about having the same server reading dozens of them from storage at once and sending them to thousands of clients around the world, the scale of the challenge quickly becomes clear. In order to best accomplish your goals, CTS recommends employing AWS S3, in addition to EC2. S3 is a storage platform designed for quick and easy access from anywhere. Storing the images in S3 allows easy replication of resources across regions so they’ll be readily accessible to players everywhere. This also means that the Java backend doesn’t need to deal with the images directly, but rather tell the clients where they are located on S3 in their region so they may be downloaded directly.  
     
   On the client-side, storage management presents less challenge, since the images are all hosted remotely. The only storage considerations would be the size of the Android / IOS apps themselves, which should be pretty negligible.
4. **Memory Management**: Within the scope of the Draw It or Lose It server-side application, memory management is largely done automatically, as part of the Java programming language’s garbage collection process. As the application runs, objects stored in memory that are no longer needed are “cleaned up” and that memory becomes available for other data. Another strategy that can be used to increase the performance of a server-side application is caching. Caching essentially means keeping some objects in memory for short periods so that they can be used to answer multiple requests, rather than having to recalculate, derive, query or fetch the same data again. This comes at the cost of using more memory on the server, but can make a big difference in the performance of some applications, especially where large files are being read from storage, or complex queries are being run against large datasets.

As for the client-side, memory management plays a critical role in the user’s experience. Large, memory-heavy applications often have long load times, and generally lack the snappy, responsive feel that a mobile application should have. In order to ensure that Draw It or Lose It provides such an experience, the client-side webapp and mobile apps will need to account for a large spectrum of devices with varying specs and capabilities, along with varying network conditions across players, perhaps even in the same game. Similar to Java on the backend, JavaScript also performs automatic garbage collection, so much of the memory management on the webapp side is taken care of there. Though, there are some opportunities to optimize the memory footprint, such as deleting the list of games after one is selected, and deleting the images from the last round once they are no longer needed. Further improvements to load-times and responsiveness can be achieved by utilizing some clever timing and delivery strategies as well. For example, rather than waiting until a game starts to request the images, perhaps they could be requested and cached locally before the game begins, while players wait for the game lobby to fill. This way, the images are ready to be displayed as soon as the last player joins and the game begins. The images for the next round could even be loaded as players are playing the first round, reducing the wait time between rounds, and ensuring a fluid, addicting experience for players.

1. **Distributed Systems and Networks**: In order to further maximize uptime and availability, Draw It or Lose It could have multiple servers in different locations, often US-East and US-West unless further subdivision is needed, to provide faster, more stable connections, distribute load, and offer redundancy in case of outages. Employing AWS services also allows for maximum scalability, to effectively serve all players at peak usage times, and also the flexibility to scale down during times when less players are online to reduce unnecessary costs.
2. **Security**: From a security standpoint, CTS recommends using HTTPS protocol for web connections and encrypting any storage medium where user data will reside. AWS offers many database options to effectively store users’ login credentials and profile information, all of which support encryption. Additionally, the EC2 instances that host the game’s components will employ firewall rules to accept traffic only on ports used for the game, and any necessary administration utilities. The Linux operating system also has impressive security capabilities, supporting many tools that may aid in further hardening the system.