

# Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 1.0 | 11/10/2023 | Brian Keller | Web-based version ‘Draw It or Lose It’ app |

**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 is looking to create a web-based version of their current android game ‘Draw It or Lose It’. In this game, one or more teams compete to solve timed word puzzles based on stock images from a library. This requires unique team names and one instance of each game.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

* Each game will be able to have one or more teams.
* Each team will have one or more players.
* Each game’s name is unique.
* Each team name is unique.
* Only one instance of each game can exist in memory.
* Project runs in web browser.

## [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 UML diagram below shows how the GameService, Game, Team and Player classes are related. The GameService class uses the singleton pattern to ensure that only one instance of each game is in memory at a time. GameService also includes functions to retrieve information about the current game. The Game class holds information on the teams in the game and includes a function for adding more. Similarly, the Teams class holds information on players for each team and includes functions to add more players. Lastly, the Player class holds functions to deliver information on players.

Each of these classes has a ‘zero to many’ relationship so that at first the GameService has zero to many games, each game has zero-to-many teams and each team has zero-to-many players. Later in the program constraints to ensure each game has the correct number of teams and players to begin can be added. The Entity class is a base class that uses information from the Game, Team and Player classes to hold attributes. The main application also uses a singleton tester to ensure that each game is kept in memory only once.

**"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** | Mac is not the first choice for a web server due to its limited hardware choices and environment. While reliable and secure, the limited hardware does not have the power necessary for scaling a web game with a large user base. Mac’s premium hardware would also accrue great cost at scale. | Linux is an open-source OS that is highly customizable and flexible. Many web server platforms operate on Linux giving developers many options. Being open source, Linux is updated often for security and bug fixes. Unless totally unfamiliar, Linux is a robust choice for server-side web apps. | Windows offers a familiar environment for most developers. Most software can run on windows allowing for high flexibility. Using windows may be more expensive due to licensing costs. .NET Framework may be a useful tool for developers working in windows environments. | Mobile devices would be a poor choice for hosting a web application server due to the comparatively weak hardware and battery reliance. When running a web-based game, connectivity and processing power are extremely important aspects that mobile devices lack in. |
| **Client Side** | Mac has a large userbase and it is necessary to develop for this OS. There will be moderate costs to developing from mac due to hardware that is more expensive than average. Developers less familiar with the Mac ecosystem may take longer due to the learning curve. Developers with expertise in Mac development will be necessary to abide by mac specific standards and provide the best experience to a large user base. Mac will require Safari specific testing. | Linux can come in endless hardware and software configurations. Development costs and time may increase due to the amount of testing needed. In contrast to mac, Linux is less likely to be used by clients for this web game. A developer with expertise in multiple Linux configurations would be necessary for allowing the web app to run well on Linux. Following standards in CSS, JavaScript and HTML will allow for compatibility across multiple browsers. | When developing for windows users, testing the application on many types of devices (touch, screen size, many types of hardware) will be necessary due to the many devices that run multiple versions of windows. There are also many windows specific features and environments such as .NET where expertise would be necessary. Following standards in CSS, JavaScript, and HTML. | Mobile devices will likely be the highest user base and thus require special attention. Developing the web app for mobile devices may be costly due to the differences between android and iOS and possibly even other mobile operating systems. Developers would need extensive experience in developing for mobile due to the varying device operating systems, device specifications and necessary focus on application performance on smaller devices. |
| **Development Tools** | Xcode is an IDE that is exclusively used on Mac that can be utilized in the creation of this web app in Swift or objective C. Xcode itself is free but distributing the app in the Appstore has annual fees. JavaScript, CSS, React and other browser specific tools will also be needed. Git for version control and collaboration will likely be used. A development team specific to Mac and iOS development would be beneficial. | JavaScript is compatible with many browsers and will be a good choice for web app development. Node.js should be used due to its scalability and adaptability, especially in a web game use case. CSS, React and other browser specific tools will also be needed. Git for version control and collaboration. A server-side development team using Linux or Windows would be beneficial. | Development for windows will likely take place in the IDE Visual studio. XBOX development for client-side development would allow for online and hardware integration with Microsoft but require developers’ licenses. As with the other platforms, CSS, React and Git are useful. A client-side development team for Linux, Windows and android would be beneficial. | For mobile development, android studio and Xcode will be likely IDEs to use for the most popular mobile operating systems. Touch interface is also paramount for mobile applications so tools like Touch.js will allow for touch development. As with the other platforms, CSS, React and Git are useful. |

## Recommendations

1. **Operating Platform**: For the overarching operating platform to run the web app Draw It or Lose It, AWS is a great choice due to its cloud nature and ability to be scaled for changing game features and business circumstances. AWS offers many different products and services, which allows for expanding the game to more systems. AWS is also able to interface with all devices through web APIs. Built in security and memory allocation makes AWS a great choice for this web application. By using cloud servers, users on any platform will be able to access the game data from distributed worldwide servers.
2. **Operating Systems Architectures**:

For holding the pictures to be distributed to players, Amazon Simple Storage Service (S3) is an inexpensive way to hold and deliver data to customers. Auto scaling memory with elastic compute will allow for costs to correlate with the number of users and allows for surges of users to be handled automatically.

1. **Storage Management**: Currently, Draw It or Lose It needs storage for the pictures necessary to play the game, and storage for the data on players, teams, points, etc. To store the pictures, Amazon Simple Storage Service (S3) would be able to inexpensively store hundreds of thousands of pictures and quickly deliver that content to users.

To store other data, Dynamo DB can be used to create and access databases of teams and player information inexpensively.

1. **Memory Management**: Managing computation resources is important when gaming. During popular times, gaming can slow down due to the number of users. By using a cloud architecture, memory management can be dynamic, and memory can be added and removed as needed. This memory auto scaling is a feature of Amazon Elastic Compute Cloud (EC2). By using virtual servers, computing power is rented and can be easily increased when needed.
2. **Distributed Systems and Networks**: Proper networking between the client device and internal parts of the game architecture is important. In order to ensure that no part of the game’s architecture is overloaded with requests, Load balancing ensures that network traffic is distributed to available targets. AWS services like Appsync can be used to record and send real time data to users for events that are time sensitive (like being the first to answer a riddle)
3. **Security**: AWS has many security features available that would be useful to the gaming room for this application. One aspect of AWS security that will be invaluable for this application is AWS Identity and Access Management (IAM). Creating roles for all developers which outline what they have access to is a great way to limit vulnerabilities. Furthermore, technologies like AWS GuardDuty can use machine learning to detect threats. Encryption is built into many AWS features. If necessary, an entire virtual private cloud can be (VPC) can be allocated to the game to ensure that all users are vetted and allowed to access the game data.

Choosing a cloud platform to host this multiplayer allows for great customizability, security and reduces costs. When looking for a solution that will be able to change with the future of Draw It or Lose It, Cloud computing allows for changes to be made quickly.