

Draw It or Lose It

# **CS 230 Project Software Design Template**

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 | 09/20/2024 | Bryan Dailey | Creation of initial document |
| 1.1 | 10/5/2024 | Bryan Dailey | Updated document to remove instructions |
| 1.2 | 10/20/2024 | Bryan Dailey | Updated Recommendations section |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room intends to expand its Draw It or Lose It game from its current Android application to a web-based version. “Draw It or Lose It” is a team-based game reminiscent of the 1980s television game “Win, Lose or Draw.” For each of four one-minute rounds drawings of clues are rendered slowly for 30 seconds and one team guesses what the drawing represents (a phrase, title or thing.) If time expires the other teams will have the opportunity to guess once within 15 seconds. Our team at Creative Technology Solutions will help streamline the development of this web version of the application.

## Requirements

* A game will have one or more teams involved.
* Each team will have multiple players.
* Game and team names must be unique
* Users should be able to check whether a team name is available
* Only one instance of the game can be in memory at once. (Unique identifiers for each instance of a game, team or player.)

## [Design Constraints](#_2et92p0)

* Cross-Platform compatibility. We will want to make sure the web application works on as many kinds of devices as possible. Creating a web app allows you to attract a large audience. Using cross platform technologies that support many device types allows us to maximize this benefit. This does limit what technologies can be chosen as we want to avoid technologies that don’t have wide adoption.
* Server Side. We will have to make sure the game server is reliable and scalable to meet potential demand. When choosing a server-side operating system and other technologies we want to make sure reliability and scalability are strengths.
* Real Time Interaction. We will need efficient communication between clients and server as timing is an important to the game. Choosing the type of sockets and methods of communication over those channels will be important as the game is played in real time with short time windows.
* Intuitive User Interface. It should be easy for a new player to pick up and use and be familiar to existing user of the Android app.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

ProgramDriver is the point on entry for the application and SingletonTester is a test class that makes sure that there is only one copy of GameService. Entity is the base class that supports a unique ID and name attributes with getter and setter methods. Player inherits from Entity. Team inherits from Entity and adds a List of players as an attribute. It also adds an addPlayer method on top of the inherited methods. Game inherits from Entity and adds a List of teams as an attribute. It also adds a addTeam method on top of the inherited methods. GameService manages the games and has attributes: games, nextGameId, nextPlayerId, nextTeamId, and service. It also has methods for getInstance, addGame, getGame, getGameCount, getNextPlayerId, and getNextTeamId. GameService has a 0 to many relationship with Game. Game has a 0 to many relationship to Team. Team has a 0 to many relationship with Player.

**"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** | Not usually used for hosting applications it is based off UNIX and can run some server applications on technologies like Node.js  Advantages would be it could be used for development as well and has strong integration with other apple platforms.  Weaknesses would be a higher cost for hardware and it isn’t common used for server applications. | Linux is commonly used for webhosting and is know for it’s stability, performance and security.  Advantages are being open-source, highly customizable and a strong support community.  Weaknesses are it does take someone knowledgeable with the platform to setup an maintain and may not be as user friendly. | Windows is commonly used in the enterprise and support a lot of web technologies.  Advantages are that if you choose other Microsoft technologies in the stack they will be easily integrated.  Weaknesses are that it comes with licensing costs and may be more resource intensive. | Mobile devices are not used often for servers. They are small efficient and self-contained with their own battery backups.  Some advantages are they would be reliable with their own built in battery backup and would be power efficient.  Weakness comparably weak hardware than a full computer and hard to scale. Mostly wireless networking. |
| **Client Side** | Mac is a less popular desktop operating system that share a lot of technology with other Apple platforms.  Advantages. A lot of the underlying work for developing a web app for iOS would likely be used when developing for the Mac.  Weakness Targeting macOS specific frameworks would be challenging with not much benefit for a small user base. | Linux is a open source platform that has a lot of variability among it’s different distributions.  Advantages supports open source software and knowledge of Linux is somewhat common. Users would likely be capable of doing some setup on their side.  Weaknesses there a many different distributions and targeting more than a few popular ones may be tricky. | Windows is a very popular desktop operating system that supports different types of web browsers.  Advantages a lot of popular web browsers available to target.  Weaknesses it could be time consuming to target multiple version of windows | Mobile devices are very common device that most people own.  Advantages large user base. Only must target two operating systems. Most modern devices will support modern technologies.  Weaknesses. Variability in screen sizes would need to be accounted for. Would have to develop with touch controls in mind. |
| **Development Tools** | Xcode, Visual Studio Code, and Eclipse are some of the IDEs available on Mac.  Languages like Java, JavaScript, HTML5, and Python could be use to make this software on Mac. You could virtualize Linux and Windows on the Mac creating a one stop platform for development. | Visual Studio Code, Eclipse and Vim are all popular on Linux. Technologies such as JavaScript, Python, PHP and Ruby can all be developed on Linux. | Visual Studio, Eclipse, and ItneliJ IDEA are IDEs that are available for Windows. Some windows specific technologies would be ASP.NET Progressive Web Apps on Windows and WebView 2 | Swift Playgrounds is available on ipadOS github.dev is available on the web. HTML and Python would be some of the languages available. While possible development on mobile would be the most tricky of the options available/ |

## Recommendations

1. **Operating Platform**: Seeing as it is very common in the industry, I would recommend that Linux be the server-side software operating Platform we use. Something like Ubuntu Server would be light on overhead and we could install a lot of open-source software if needed.
2. **Operating Systems Architectures**: Choosing a popular version of Linux like Ubuntu Server allows us to take advantage of a modular design and only install the software that we need to run our software. Linux would support a client-server model allow our players to connect to a centralized server.
3. **Storage Management**: We could utilize something like Network File system to store data related to the software if we need multiple servers to access the same data files. A central storage system like this could be used to help ensure uniqueness of names of teams and players across servers.
4. **Memory Management**: Linux allows for swap memory. This would allow us to utilize less actual RAM and substitute cheaper storage for games that are paused or finished.
5. **Distributed Systems and Networks**: We can use WebSockets or similar technology to interface clients with the server. We can also have many servers for resilience to handle things like network outages.
6. **Security**: We can use SSL/TLS to encrypt client data on its way to and from the server. We could implement user authentication to make sure only the player can sign into their account. Security updates are regularly released for popular Linux platforms and keeping the server up to date will be important.