

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 10/20/2024 | Carolyn Rios Borges | Design Constraints for Draw It or Lose It. |

**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)

Draw It or Lose It, a web-based game in development by The Gaming Room (our client), is currently available as an Android application only. Draw It or Lose It encourages teams to compete and guess what is being drawn. Instead of requiring the players to draw an image in turn, the application renders images from a large library of stock drawings. Drawings are rendered steadily and completed in 30 seconds. A team has a minute (one minute per each round, 4 rounds in total) to guess the puzzle, and the remaining teams will have 15 seconds and one chance to guess the puzzle if the team time expires.

The software design problems that our client is currently facing is the application availability only for Android users and their staff’s inability to set up the environment. Additionally, the client’s software requirements dictate the need for multiple players and teams, which would require more than just a singular Android application to be developed. I therefore propose a transitioning the development to a web-based technology and language, which is better suited to the client’s requirements. This will focus the team’s development efforts and allow for compatibility beyond just the Android platform.

## Requirements

[Req-1] Each game will be capable of having the involvement of one or more teams.

[Req-2] Multiple players can be assigned to each team.

[Req-3] Each game or team name is unique, and names currently in use cannot be taken.

[Req-4] In memory, only one instance of the game can exist, which can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

[Const-1] Due to technological limitations, each game can only handle 4 teams per each gameplay.

[Const-2] As the developer, I would have to develop in a web-based technology.

[Const-3] The application would require network connectivity.

[Const-4] The application would require players to sign up.

## [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 class diagram below contains the contents of a class, which is also known as the interface. It also contains the relationship between classes, which is the most important detail of this UML class diagram. As we can see, the ProgramDriver class uses the SingletonTester class, and vice versa. Game, Team, and Player classes inherit from the Entity class. By inheriting from the Entity class, we reduce code duplication. Game service can contain 0 or more games, teams, and players. In other words, it takes many players to form a team; it takes many teams to form a game, and all games can be managed by a single service.

**"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** | For Mac clients using software that is inherently compatible with the MacOS desktop distribution makes for easier management and interoperability. But, as a con, Apple no longer offers an Apple Server OS, with their offering being discontinued as of April 2022. | Linus is the most common server OS distribution in use. It in enjoys wide community support and active security updates from numerous parties. As a con, management and administration is sometimes more difficult as it often takes an experienced system administrator to configure and solve problems within the system. | Windows is the most popular desktop distribution by a large margin. This has the advantage of great compatibility with their server distribution. Management and administration of both OS and desktop can be centralized and can less skill to manage. As it’s the most popular, it is often the target of hackers and spam artists, seeking to compromise and steal private data. | For ad-hoc setups, a mobile device can be perfect. Particularly when the number of concurrent users is very small. However, given the slow speed by comparison to a server, this means that a mobile device will never be able to manage or administrate hundreds of users at the same time. Additionally, heat may be an issue, as small mobile devices are not meant to process at 100% for long periods of time. |
| **Client Side** | Assuming our application is Mac native, the cost of a license may be prohibited in order to deploy it. Given the small number of Mac OS compared to Windows users, it may not make sense to create a native application for this platform. However, going native would fit in with the existing OS user interface and provide a more polished look than alternate methods. | Linux is preferred by software developers due to the ability of having full control. You can change the look and feel of the application. Additionally, as most of the underlined libraries are open-sourced, it is easier to include and debug them, if necessary, which might not be the same with Mac or Windows OS. However, the Linux desktop experience is fragmented across multiple desktop environments and distributions. Developing for one, at best, wouldn’t match the look and feel of another. And, at worst, wouldn’t work at all outside of the OS it was built on. | As the largest user base of any OS, many of the libraries and frameworks that we are expected to use, are certainly guaranteed to work on Windows. Items such as hardware acceleration and multi-monitor support are far less painful to configure and utilize within Windows, than it is on any other platform. As a con, depending on language, time investment in Windows may be significant, and may lock any applications to the platform in terms of compatibility. | Has the largest user base next to Windows. Millions of devices are sold and utilized each year, making mobile devices a necessity. Allows for application use on the go, and away from the traditional desktop. As cons, the market is currently split between Android and iOS. Meaning development for one will prevent its use on the other. Screen size and layout will also present a challenge which will increase development time. |
| **Development Tools** | Swift, Objective C, C++, JavaScript, VS Code, Xcode, HTML, CSS, React, Python, etc. | Visual Studio Code, Vim, C++, C, React, Rust, JavaScript, Python, HTML, CSS, etc. | Visual Studio Code, Visual Studio, C Sharp, Visual C++, VV.net, Vim, C++, C, React, Rust, JavaScript, Python, HTML, CSS, etc. | Android Studio, Xcode, Swift, JavaScript, Python, etc. |

## 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**:

It is a requirement that the game Draw It or Lose It be cross platform in nature. This means that it needs to work on mobile hardware (such as phones and tablets) as well as desktop computers. Therefore, we recommend using web technologies such as React or Node.js for the client interface, which will enable the game to be loaded on any browser implementation.

It is also recommended that the backend service that contains the core game logic, assets, and

user stats, be written for enterprise server hardware running an Linux OS. This will provide the necessary processing power to support many concurrent users.

1. **Operating Systems Architectures**:

For the client application we recommend using web technologies such as React or Node.js, just as stated in item 1 above. These technologies can be run in any web browser and are available under all major operating systems.

For the backend service we recommend targeting a Linux Operating system. Linux gives full control to software developers, and contains various libraries that are open-sourced, making it is easier to include and debug them, if necessary, which might not be the same with Mac or Windows OS.

1. **Storage Management**:

A Cloud Server would be an appropriate choice as this provides a seamless experience for all players and reduces the end user requirements. All game assets, user stats, and player lobbies would be stored here. The game client therefore only needs to connect to the server and load any assets served to it.

1. **Memory Management**:

We recommend utilizing web caching techniques for service assets and stats. If necessary, a CDN could be employed to better serve players on a national level. Web browsers can cache previously visited pages which further reduces the need to download all the game assets at every login.

1. **Distributed Systems and Networks:**

The backend service should be run on a cloud provider that has regional data centers and service mirroring. This means that the service can be run in multiple places across the world and allow players to connect to their local instances. Each service would track their user progress and share that progress with all the other services. This creates a system whereby clients depend on services being available, the network being reliable, and the connection between services consistent to avoid any data duplication.

1. **Security:**

No core logic should be present in the client application. This prevents the user from tampering with any win conditions or accessing any other player information. The client application should be responsible only for rendering the game on the client’s browser and transmitting actions to the central service.

On the target platforms, the client should run in user mode and not need access beyond system memory and cache storage space. The client should disallow running with elevated privileges to prevent any unforeseen attacks.

All communication and data should be encrypted at rest and in transit. SSL should be employed for all communication and an appropriate encryption algorithm chosen for high consequence user data such as emails and passwords. Users should be required to register with the game service and given a unique ID. This id would be used to tag all user actions so that anything nefarious can be later tracked and the offender banned.

Last but not least, the cloud service should be properly configured and managed to prevent unauthorized access.