

Draw It or Loose 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 | 9/22/2024 | Eric Pelletier | First Draft |
| 1.1 | 10/6/2024 | Eric Pelletier | Second Draft added client side, development tools, and server side |
| 1.2 | 10/26/2024 | Eric Pelletier | Recommendations section updated |

**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)

This software design document is made to develop the application Draw It or Lose It for The Gaming Room. It is based on the 1980s television game Win, Loos or Draw where teams compete to guess what is being drawn. The application will render images from a large library of stock drawings as clues. Drawings are rendered at a steady state rate and fully complete at 30 seconds. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle withing 15 seconds.

## Requirements

1. *Game will have the ability to have one or more teams involved*
2. *Each team will have multiple players assigned to it*
3. *Game and team names must be unique to allow users to check whether their name is in use when choosing a team name*
4. *Only one instance of the game can exist in memory at any given time.*
5. *The game should have specific time limits drawings rendered at 30 seconds. If team does not guess puzzle remaining teams have 15 seconds to answer*

## [Design Constraints](#_2et92p0)

1. Single Instance Limitation: The design must account for the limitation of only one istance of the game service active in memory at any given time.
2. Web based environment: The game application is a web based. Security, and compatibility with different web browsers is top priority.
3. Unique Names: Player names, team names, must be unique to avoid conflicts.

## [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 shows the classes that are in the Draw it or Loose it game. The Entity class is the base superclass for all the entities in the game. Its attributes being id and name that are shared by all the subclasses. Game, Team, and Player classes are subclasses that extend the entity superclass.

The GameService class consists of attributes nextGameID, nextPlayer, and NextTeamid. It has a composition relationship with the Game class. The Game class has a composition relationship with the Team class and has a composition relationship with the Player class.

The ProgramDriver class is the main(). It is responsible for adding games, teams, and players using the instance of the GameService.

The diagram shows the inheritance between the Entity superclass and its subclasses which is Game, Team, and Player. Inheritance lets subclasses inherit the attributes and behaviors of the superclass. The GameService class encapsulates the attributes, the constructor and the list of existing games, and has methods to interact with the data. This ensures only on instance is created and promotes abstraction.

**"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** | Macs are more expensive for hardware.  Unix based. Very stable and secure for web applications. Limited scalability compared to Linux or Windows.  An advantage is it has the ability to run MacOS, Windows and Linux apps. MacOS is very consistent and easy to operate | GUI limitations as well as hardware compatibility. Open source operating system. Scalable very stable and secure.  Linux well know for versality in servers and has superior security protocols compared to windows. Disadvantages are lack of choices for prebuilt machines and file format compatibility. | Very strong software compatibility. But windows is know for being vulnerable for security. Windows based authentication integrates with active directory based on corporate servers, no addition expenses needed | Touchscreen as well as limited screen size. Very portable.  Can be beneficial if the user are limited and the application isn’t too large or complicated. Disadvantages is mobile based hosting is cloud based which cant make it more vulnerable to hackers. |
| **Client Side** | User friendly. To develop and maintain clients will increase development costs, time and expertise. It is not available unless on an apple device. | Free to use. Diverse expertise needed for different clients.  Linux is free and open source it is easier to maintain and upkeep. Security is an issue compared to MacOS or Windows | Licensing costs very high compared to other alternatives. Windows offers hardware support.  Someone needed with Windows OS expertise and costs depending on specific features. | Connectivity limitations. Has push notifications, camera, and GPS. Must able to operate on mac and android. Mobile devices have many different OS each phone is geared toward that OS. |
| **Development Tools** | Node.js, JavaScript. VSCode and Xcode. | VSCode Atom, SublimeText.  Apt and yum | C and .NET  Visual Studio, Jet Brains | Swift, Objective C, Java and Koltin. Android studio, xcode. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.> Linux based server would be the best. Linux server decreases the license costs and doesn’t limit access to data centers like Windows. Front end will be agnostic to the backend and connect APIs. Linux offers good security and operability. There are many tools available including security software because Linux is the most common server platform. Front end will be written in SWIFT for iOS, Java for android, .NET for Windows.
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.> The best architecture would be backend server that manages the game environment and frontend/client based rendering. Draw it or Lose it game doesn’t need low latency between the frontend and backend. Transmissions can be done asynchronously. A backend running microservices with Docker could allow for scalability. Before an architecture was developed a cloud provider would have to be selected. Some providers have proprietary tooling provided to you. Frontend rendering lets the server offload some resource intense parts of the Draw it or loose it app and reduce monthly data center costs. Client side rendering should insulate gameplay from network issues because framerate is important to the overall gameplay. The client should cash some of the images so that there is a smoother gameplay experience and rendering. Doing a brower based game with PWA would be the best choice.
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.> HDDs or SSDs should provide the performance needs of the application. Providing cashing behavior and client side rendering. If the Game Room purchases their own hardware then no decision on storage medium should be made. Cloud native tools will add flexibility on the server side. This was the scalability or localization concerns are taken care of.
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.> Linux uses pagecache for data stored in main memory, virtual memory for any pages that are allocated. Linux uses demand paging which helps to give lower memory usage. Pages not actively being used wont be loaded into memory. Least Recently Used algorithms will decide on page replacement. The Android Runtime and Dalvik virtual machine uses paging memory mapping to manage memory. Whatever memory the app modifies if it was allocation new objects or touching mmapped pages it will remain resident in RAM and will not be paged out. Memory management iOS wasn’t initially non=ARC where it retained and released objects. But now it supports ARC and we don’t have to retain and release the objects. Xcode will automatically take care of it when it compiles. Minimum RAM will be needed on the server side will be needed with client side rendering. Client side RAM should be minimum because only a couple of images need to be stored in memory at any given time.
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).> Uptime and outage prevention is the biggest reason so many applications are built in cloud native architectures. Cloud providers can replicated and changed services with different deployments to prevent outages. The front end and backend will communicated through RESTful APIs asynchronously. RESTful API usage lets the client/server communication be transparent to the deployed frontend.
6. **Security**: This web based game should implement secure communication. For users there should be authentication and authorization. Usernames and passwords, as well as at least three security questions. An email account should be added for password reset or if username is forgotten. Encryption should be used to safeguard sensitive information and users personal information. Security should consist of Role-based authorization. Also the least privilege should be employed which limits users in their game controls, game creation, team creation, and team enrollment. There should be a team captain who can add or remove players from a team. No users in the game should be granted ADMIN on the system. Also a firewall should be added as part of the server.