

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.j2v47soph4ry)

[**Table of Contents 2**](#_heading=h.yqv95pkgck3q)

[**Document Revision History 2**](#_heading=h.dukaw8h6s43d)

[**Executive Summary 3**](#_heading=h.6heien1490gc)

[**Requirements 3**](#_heading=h.3mplnveftnzt)

[**Design Constraints 3**](#_heading=h.8c085hn8g71n)

[**System Architecture View 3**](#_heading=h.s53n0gecznzp)

[**Domain Model 3**](#_heading=h.marafboywpo0)

[**Evaluation 4**](#_heading=h.ypm2fmccy5gh)

[**Recommendations 5**](#_heading=h.jjezy1u38uh8)

## [Document Revision History](#_heading=h.ewofyl22sln3)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 7/20/25 | Aaditya Fadnavis | Contents Updated |
| 2.0 | 8/3/25 | Aaditya Fadnavis | Contents Updated (Evaluation and Below) |
| 3.0 | 8/11/25 | Aaditya Fadnavis | Contents Updated (Recommendations) |

**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.bla4n4zxwbt)

The client, The Gaming Room, is looking to develop a web-based game that can run on multiple platforms. The game will be based on their current game, *Draw It or Lose It*. This app is currently only available on android but is planned to be developed in ios later on. The application will include images being rendered from a large library of stock drawings and so will require appropriate handling for each platform. The game consists of four rounds with each lasting a duration of one minute. After 30 seconds, the image is fully rendered and if not guessed correctly by the current team, a chance is given to others consisting of 1 guess within 15 seconds.

Furthermore, the game will be required to be capable of involving one or more teams with each team having multiple players within. No duplicate names will be allowed for the game and team members as well. Specific functions will be used to verify that no duplicate names exist. Along with this, only one instance of the game will be allowed in memory at any time. Unique identifiers will be created for each instance of a game, team and players so that it can be more efficiently verified.

Currently, only softwares will be developed which will be followed by hardware developed based on the current software, in the future. Because the game will be released on multiple platforms besides just android, the code to be developed will take into account the specifications for each platform and use coding best practices appropriate for each. This will maximize user ease and software efficiency also will make sure time and energy is not used unnecessarily in slower and buggy processes.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_heading=h.21db8ny2td96)

1. **Singleton Game Instance Constraint**: Only one game instance can exist in the memory at any given moment. Implications of the design constraints on application development are that the SingletonTester class will verify that only one game service exists. In a distributed environment, the use of distributed caching or locking to enforce singleton behavior will be needed along with a careful synchronization of state between instances if scaled horizontally.
2. **Multiple Teams and Players per Game**: A game must have one or more teams with each team having multiple players. Implications of the design constraints on application development are that relational or hierarchical data structures will be needed to manage parent-child relations. In a web app specifically, backend endpoints will be required to create teams, add players, and join games along with real-time coordination to reflect team status updates, player joins, and more.
3. **Unique Game, Team, and Player Names**: Names must be unique and verifiable before a game creation to avoid duplication and associated problems. Implications of the design constraints on application development are that the backend must be queried to check availability before finalizing creation and a centralized database or index for fast search must be used. Race conditions will also need to be avoided by using atomic operations or mutex locks for scenarios where multiple users create entities at the same time.
4. **Timed Game Logic & Drawing Phases**: Each round lasts for a duration of 60 seconds with the first 30 seconds for drawings completion and 15 more seconds for other teams to guess when the main team fails. Implications of the design constraints on application development are a game loop controller or scheduler that will control the timer, send partial drawings over time, and start a 15 second timer for other guesses when the main team fails. Additionally, drawings may need to be delivered in parts for progress to be seen in the drawing as the 30 seconds go by.
5. **Web-Based Distributed Deployment**: The application is deployed in a distributed, web-based environment including android. Implications of the design constraints on application development are the incorporation of a stateless frontend and stateful backend along with distributed databases and a shared memory cache for session persistence and gameplay data persistence. Functions may also be needed to make sure the game is recoverable.

## [System Architecture View](#_heading=h.fiu9klt3hq7z)

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.q2ewgpl5rk7x)

The UML diagram below successfully illustrates the multifaceted nature of the program being developed. The various classes and their relationships can be seen. For example, the Entity class has private attributes and one private and rest public methods that are inherited (shown by the hollow triangle line) by the Game, Team, and Player classes. Additionally, each of these classes also have their own attributes and methods as well as relationships with each other. We can see this from the plain association lines between Game, Team, Player, and GameService classes where each class has a none to many relation (0…\*). Specifically, GameService class is associated with its parent Game class, the Game class has an inheritance relation along with an association relation to the Team class, the Team class has an inheritance relation and association relation with the player class, and finally the Player class has one relation which is inheritance from the Entity class. Another group is the Program driver that runs the main program and uses the SingletonTester class. One point to note is that this relation between the ProgramDriver class and SingletonTester class fulfills one of the main requirements of having only one instance of the game in memory at any time. This is done by the SingletonTester class using the public method testSingleton() to call the public getInstance(): GameService from the GameService class and check that every reference is the same. Altogether, we can see that the UML diagram below coherently displays the relation and functions between each of the classes in the aforementioned game.

**"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.q4uz4bgksh88)

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** | macOS supports web servers such as Apache, Nginx, and tools like Docker or Node.js, but it is uncommonly used in production due to licensing and cost complexities. It is ideal for local development but less scalable than Linux. Licensing costs for macOS servers are tied to Apple hardware, which can increase the overall server expense. | Linux is the most common server OS. It is fast, stable, and supports all major web technologies. It is also free, cloud-friendly, and ideal for hosting, but it may require more command-line skills. Linux is open-source and free to use, making it highly cost-effective for server deployment. | Windows servers support IIS and .NET well but are heavier and more costly as well as more complex. They are better for Microsoft-based stacks, but less efficient than Linux for open-source tools. Windows Server requires licensing fees, which can increase the total hosting cost for The Gaming Room. | Mobile devices aren’t suited for hosting web apps due to limited power, storage, and uptime. They are not practical for backend roles in production. Mobile devices are not used as servers, so there are no operating system licensing costs. |
| **Client Side** | Supporting Mac clients requires Apple hardware and some platform-specific UI work. Development is costlier too due to tooling and testing needs, but important for reaching Mac users. | Linux desktop clients are rare for games. Browser-based support is enough for most. Native support isn’t worth the cost unless targeting developers. | Windows dominates desktop usage. It is essential to support and easy to develop for. Tools are mature, and developer availability is high. | Mobile support is critical. Native (Swift, Kotlin) or cross-platform (Flutter, React Native) tools are needed. It increases cost and time but expands reach. |
| **Development Tools** | Mac supports Xcode, VS Code, Docker, and backend languages. It is required for iOS apps and has great UNIX tools for full-stack work. Most Mac development tools like Xcode and VS Code are free, but some third-party tools may require licenses. | Linux is ideal for backend work with tools like VS Code, Git, and terminal utilities. Most web tools run smoothly and natively. Development on Linux mostly uses free and open-source tools, minimizing licensing costs. | Windows supports all major IDEs, with strong .NET support. WSL helps run Linux tools. It is versatile but sometimes needs extra setup for open-source tools. While many Windows tools are free, some like Visual Studio Enterprise or proprietary software may require paid licenses. | Mobile developers use Android Studio and Xcode. Cross-platform tools like Flutter or React Native can also help. Setup is more complex, but necessary for app store deployment. Mobile development tools are mostly free, but publishing apps may involve platform-specific fees (for example the Apple Developer Program for Apple). |

## 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**: Windows is the best choice for The Gaming Room’s *Draw It or Lose It* application because it has a wide adoption, strong development tool ecosystem, and compatibility with many enterprises and consumer devices. It supports both web and desktop environments, offers good integration with .NET technologies, and allows for rapid development using Visual Studio. Additionally, Windows Server environments are capable of hosting scalable web services, while Windows desktops provide an easy platform for local testing and administration.
2. **Operating Systems Architectures**: Windows supports both 32-bit and 64-bit architectures, but the 64-bit architecture is recommended for modern development and scalability. Windows architecture is modular, allowing the separation of the application layer, system services, and kernel. It supports multithreading and asynchronous operations, which are important for real-time games similar to *Draw It or Lose It*. Its architecture also supports Windows Subsystem for Linux (WSL), which can be helpful when integrating Linux-based development or testing environments.
3. **Storage Management**: Windows uses the NTFS (New Technology File System) as its default, which provides support for large volumes, encryption, and access control. For server-side storage of game data, player progress, and media assets, Windows can work with Microsoft SQL Server or cloud storage solutions such as Azure Blob Storage. NTFS combined with cloud integration offers a flexible, secure, and scalable solution for managing static assets and user-generated content.
4. **Memory Management**: Windows employs a virtual memory management system that includes paging, memory prioritization, and caching. It dynamically allocates memory based on process needs and supports garbage collection through frameworks such as .NET, which is useful for managing the memory footprint of the application. Tools like Task Manager, Resource Monitor, and Visual Studio Diagnostics help developers identify and resolve memory-related issues during development and deployment quickly and efficiently.
5. **Distributed Systems and Networks**: To support cross-platform interaction, web services built with .NET Core or ASP.NET, hosted on Windows Server or in the Azure cloud can be used. These services can expose RESTful APIs or WebSockets for real-time gameplay communication. A distributed architecture should include load balancers, failover mechanisms, and distributed caching to manage state and support horizontal scaling. Proper network configuration, redundant pathways, and health checks can help maintain uptime and responsiveness during possible partial outages or heavy traffic.
6. **Security**: Windows provides strong security features including user authentication (via Active Directory or OAuth), encrypted file systems, Windows Defender, and firewall configurations. To protect user information, HTTPS/TLS should be enforced across all endpoints. Data should be encrypted at rest and in transit. Microsoft Identity or Azure AD B2C can be used to ensure secure user sign-in and access control. Regular updates and patching, role-based access control, and audit logging can help maintain security compliance and user trust across all platforms.