

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** Error! Bookmark not defined.](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/17/24 | Jermaine Wiggins | Initial software design recommendations |
| 1.1 | 12/1/24 | Jermaine Wiggins | Evaluation of various operating systems |
| 1.2 | 12/15/24 | Jermaine Wiggins | Recommendation of operating system |

## [Executive Summary](#_sbfa50wo7nsh)

The gaming room is requesting our team facilitate the development of their popular gaming app Draw It or lose it to a web-based cross platform version. To create a seamless web-based application that meets the clients’ needs the team must ensure the following, support multiplayer through teams with unique games and team names, while always having only one instance of a game in memory. To achieve this the proposed solution is to use a singleton method (for the single instance of a game in memory at a time) and an iterator (to manage and ensure game and team names are unique).

## Requirements

* Support single or multiteam games
* Each team will have multiple players assigned to them
* Game and team names must be unique
* Only one instance of the game can exist in memory at a time

## [Design Constraints](#_2et92p0)

* Single game instance: Because this is a requirement, the singleton pattern must be used while ensuring duplicate instances aren’t created
* Cross platform compatibility: The web-based application must be compatible with the mobile version and preform the same on all browsers.
* Resemble app counterpart: replicate mobile app features as closely as possible but some features may not be possible or require an alternate design
* Budget constraint: though not mentioned budget is always a constraint and depending on the set amount will determine if prioritizing advanced features over core optimization is practical.

## [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 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.**

* The Program driver class uses the Singleton tester class to test if there’s is in fact one instance of game established by the game service class through a singleton pattern.
* Entity is the parent class to the game, team and player class which they all inherit attributes and methods from entity. Reducing code duplication and representing the OOP principal inheritance. Additionally, not including things like name and id in teams and game allows the classes to display more essential properties representing abstraction.
* Encapsulation is displayed by the entity, game, team, player and game service classes. Each have private attributes that can’t be modified outside the class, but only through setters and getters.
* Each child class, game team and player class override the inherited method tostring and implementing it in its own way, representing the OOP principle of polymorphism.
* Game service class through the add game method with an iterator ensure game names are unique
* Game class through the add team method with an iterator ensure team names are unique
* The association relationships between the game, team, and player classes ensure that a game can have multiple teams, and each team can have multiple players fulfilling the requirement of handling single or multiple teams and players in a game.
* The game service class is associated with game class and there can be 0 instances to many respectively.
* The game class is associated with the team class there can 0 games and many teams
* The player class is associated with the team class there can 0 teams and many players

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * Seamless integration with reliable apple devices and hardware (Macs iPhone). * Bulit in security features to protect against malware. * Hardware is more expensive (must have mac to develop mac applications) * Not compatible with many non-apple specific applications * Doesn’t support a native macOS server (apple no longer supporting, so harder to scale with larger player base). | * Open source and free to use (no licensing cost). * Customize based on needs (limited to the user expertise) * Optimized to handle high traffic workloads(scalable). * Built in firewalls to protect against common vulnerabilities * Large learning curve if not familiar with interface. * Many Linux distributors so it may be challenging choosing best option if not familiar. * Support to troubleshoot could be costly if not familiar | * Very user-friendly built-in GUI vs command line interface. * Easy integration with other Microsoft tools like azure to scale applications easier. * Sever requires licensing (cost depends on the edition) * Higher amount of security vulnerabilities | * Not typically used to be a server / not a direct hosting platform. * Limited hardware like processing power and storage. * Licensing cost depends on cloud service used. |
| **Client Side** | * Free developer tools, but dev program fees (Appstore). * High cost and complexity added development time to integrate cross platform tools. * Expertise needed in macOS tools, apple interface guidelines, and cross platform frameworks like electron. | * Open source so none to low development and licensing cost. * Expertise in Linux based tools. * Supporting multiple clients can increase development time and knowledge on integration tools like Electron. | * Licensing cost (windows 10/11) and development tools (visual studio) cost. * Requires expertise in windows development tools, APIs and UI guidelines. * Requires expertise in cross platform frameworks like electron and react. | * Licensing fee for android and apple $25 one-time fee and $99/year respectively. * Needs expertise in android and iOS development tools and app guidelines * Needs expertise in cross platform development tools like react. |
| **Development Tools** | * Programming languages: Swift/ objective C (mac and iOS) JavaScript HTML/CSS (cross platform) python (backend development) * IDE: Xcode (for mac development/ deployment) Visual studio code for (cross platform) * Cross platform integration: electron (desktop app) React (mobile app) | * Programming languages: C++ / python (backend) JavaScript(frontend) * IDE: Visual studio code can develop numerous languages and frameworks. * Cross platform integration: React (mobile app) Electron (web-based applications windows mac and Linux) | * Programming language: C# and C++ and python (desktop and web)   JavaScript/CSS (front end)   * IDE: Visual studio (windows native)   Visual studio code (cross platform works with many languages).   * Cross platform integration: React for mobile development and Electron for cross platform web applications. | * Programming language: Swift (apple) java(android).   JavaScript (cross platform development)   * IDE: Xcode(apple)   Android studio/ Android SDK (android) Visual studio code for cross platform development   * Cross platform integration: React |

**Recommendations**

**Operating Platform**

Based on the information gathered regarding possible operating systems, the team recommends Windows for the Gaming Room to expand Draw It or Lose It to other computing environments. While Windows has some drawbacks, such as being more vulnerable to security attacks, its advantages far outweigh these concerns. Windows provides a user-friendly interface, compatibility with cross-platform development tools (such as React, VSCode, and Electron), built-in tools on Windows Server to support scalability, seamless integration with its native cloud service Azure, and is the most widely used operating system, ensuring community support.

**Operating Systems Architectures**

Windows core design is providing a user-friendly experience, it achieves this by dividing the OS into two modes, user mode and kernel mode. User mode is where the user interacts with the UI, such as running applications, but doesn’t have full access to hardware or system resources, ensuring system stability and security. While kernel mode has full access to hardware and system resources, and task such as handling all the I/O operations, manages memory, and the file system. Kernel also mode ensures that the operating system can efficiently manage processes, allocate system resources, and handle critical system-level tasks like defining how data is stored, organized, accessed, and deleted.

**Storage Management**

Draw it or Lose it is an extremely popular game that is expected to continue to grow rapidly once cross platform integration is available, because of this storage needs to be able to scale and grow with the application. An appropriate storage management system would be a native cloud based one, like Microsoft Azure. Not only does Azzure offer a scalable storage solution, its more cost effective, data is easily accessible (also backed up with built in recovery features, making it reliable and secure), seamless integration with other cloud-based services like azure app services, azure sql database (organize structured data), azure files (future file sharing needs) and azure devOPs (deploy updates).

**Memory Management**

Windows has built in features for optimizing memory management, like virtual memory management, memory allocation, dynamic memory scaling and automatic garbage collection. Virtual memory is used to allow the game to use more memory than what is physically available by swapping memory pages to the disk, freeing up processing space, this is useful if the system is under heavy load. For memory allocation windows uses a memory manager to allocate memory resources for the game’s processes ensuring sufficient memory is available to handle all tasks. Dynamic memory management enables the windows to allocate more memory during peak traffic times and release memory when demand decreases. Automatic garbage collection reclaims memory from unused to prevent memory leaks, which prevents excess memory consumption as the game is running.

**Distributed Systems and Networks**

Distributed software will be built on a server client architecture, with a server handling processes while the game runs on various platforms (web and mobile devices). RESTful APIs allow for communication between the game client and the servers, such as allowing the client to send request to the server like get user ID. Dependencies between client and server include the server handling game flow and managing player data which would depend on our suggested cloud storage. Each game clients connect to the server independently but depends on the server to receive and send data. To handle connectivity issues or outages caching can be used to store some game data on the device or in the devices memory to allow for the game to run locally temporally until connection is restored, ensuring smooth gameplay during network disruptions.

**Security**

Despite being the most susceptible to security attacks, windows provide built in features like built in firewalls (windows defender) and azure recovery tools to prevent minimize these risk. Other ways to ensure information is protected is through data encryption, protecting RESTful APPIs, multifactor authentication, and role-based access. Data encryption is used to ensure sensitive data on the server and in transit are encrypted. When multifactor authentication is implemented only authorized user can gain access to sensitive information. Finally, role-based access restricts what different users has access to.