

**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 | 01/26/2025 | Liliana Jimenez | Executive summary, design constrains and explanation of UML diagram. |
| 1.1 | 02/09/2025 | Liliana Jimenez | Evaluation; server side, client side, and development tools. |
| 1.2 | 02/23/2025 | Liliana Jimenez | 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](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based version of Draw It or Lose It, a multiplayer game where teams compete to guess puzzles based on visual clues. The application must support multiple teams and players, ensure unique names for games and teams, and maintain a single instance of the game in memory.

## *Problem*

The main challenges are managing unique game and team names, supporting multiple players and teams, and ensuring only one game instance exists at a time. Additionally, the web-based environment introduces constraints like concurrency, scalability, and low-latency requirements.

## *Solution*

* Singleton Pattern: Ensures only one GameService instance exists to manage the game.
* Iterator Pattern: Prevents duplicate names by iterating through collections.
* Encapsulation & Abstraction: Simplifies entity creation and management.
* Inheritance: Reuses code via a shared Entity class for games, teams, and players.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

1. **Unique IDs and names**: Every game, team, and player need a unique number or code to tell them apart and avoid confusion.
2. **Only One Game Instance**: There should only ever be one version of the game running at a time, which can be controlled using a Singleton pattern.
3. **Teams and Players**: Each game can have several teams, and each team can have many players. The app needs to organize these relationships carefully.
4. **Web-Based App**: The game will run on the internet and needs to work well across multiple servers, handling lots of users at the same time.
5. **Drawings**: The app needs to show clues as drawings that slowly appear over time, finishing by the halfway point of the round (30 seconds).
6. **Timed Rounds**: Each game round lasts one minute, with strict timers for guessing, so the app needs accurate timers that work across all users.

**Implications**

1. **Scalability**: The app needs to handle more users, teams, and games as it grows, which might mean using cloud services or dividing the system into smaller parts that work together.
2. **Performance**: The app needs to have low latency. Features like checking names or loading drawings need to be fast and smooth, even when many people are playing.
3. **Reliability**: Game progress, like scores or timers, should be saved properly so it is not lost if something crashes or the system restarts.

## [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 application uses a main driver class to create games, teams, and players, managed through the GameService class, which implements the Singleton pattern. This ensures only one instance of GameService exists, created via the getInstance() method, which checks if the instance is already initialized.

The addGame(), addTeam(), and addPlayer() methods use the Iterator pattern to ensure unique names for games, teams, and players. If no match is found, new objects are created and added to their respective lists.

The Game, Team, and Player classes inherit from the Entity superclass, which includes protected attributes id and name. Null objects are blocked through a protected default constructor, enforcing valid initialization.

The OOP principles the design shows:

**Encapsulation:** Controlled access to attributes and object creation.

**Abstraction:** High-level methods (addGame(), etc.) hide implementation details.

**Inheritance:** Shared attributes and behaviors via the Entity class.

**Polymorphism:** Overloaded constructors in the Entity class.

**"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** | Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Linux  and Windows  Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Linux  and Windows  Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Linux  and Windows  Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Linux  and Windows  Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Linux  and Windows.  Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Li  Mac servers, built  on Unix, provide a  stable platform for  web apps and offer  server-based  deployment  options. Despite  being developer-  friendly with tools  like Xcode, they  come with higher  hardware costs  (around $2500) and  an additional $500  for server licenses.  However, scalability  is somewhat limited  compared to Li  Mac Servers (Unix-based): While they offer a stable platform and developer-friendly tools like Xcode, their high upfront cost (around $2500 for hardware, plus $500 for server licenses) makes them a more expensive option. Scalability can be limited compared to Linux and Windows. They are best for specific environments, like app development or small-scale operations, but the hardware costs can make them less attractive for large-scale deployments. | Linux Servers: The most popular choice for web hosting, thanks to its open-source nature, Linux is often favored for cost efficiency. The wide availability of cloud hosting options and support from major cloud providers like Google and Amazon make it a flexible and scalable choice. It's commonly used for hosting websites at an affordable price point ($9.99/month to $399/month). Its lower maintenance and licensing costs also make it ideal for startups or businesses looking to optimize their hosting budget. | Windows Servers: Known for strong software compatibility and a vast developer ecosystem, Windows servers offer excellent hardware support and comprehensive documentation. They can be a good choice when specific software or applications that require Windows are part of the project. However, the downside is their tendency to have more security vulnerabilities compared to Linux and the higher operational costs, especially for larger-scale deployments. That said, they often don’t come with service fees, making them a cost-effective choice for certain use cases. | While using mobile devices as web servers is not widely known and may seem impractical, it is still possible to implement. Oracle is one of the companies that provide mobile server-side solutions. Oracle Database Mobile Server allows the management of applications, users, devices, and data across large-scale deployments of mobile or remote devices. The key advantages of Oracle’s database structure include support for iOS and Android development tools, as well as synchronization with Oracle NoSQL databases.  Additionally, there is a spectrum of hardware capabilities across different devices, leading to variations in performance and features. As a result, their cost can vary significantly. |
| **Client Side** | Mac’s user-friendly interface simplifies learning, but developing for macOS can be costly and time-consuming, requiring expertise in Objective-C or Swift. A Mac with the latest Xcode is necessary for development. Additionally, with Windows holding 75% of the market compared to macOS’s 16%, the potential user base is smaller. | Developing on Linux is relatively straightforward since commonly used languages like Java, C/C++, and Python are viable options. Additionally, the GNU/Linux platform supports multiple users, making it suitable for server environments and multi-user applications. However, its limited adoption in mainstream consumer markets reduces the overall value of Linux development. Despite these challenges, Linux offers key advantages, including strong security, open-source flexibility, and high performance for server-side applications. | Windows development is commonly done using C# or .NET, making it accessible with no significant entry barriers. Since Windows XP, the OS has supported multiple users. With 75% of computer users preferring Windows, it presents a strong business opportunity. | Mobile devices are built for single users, but developing Android and iOS apps is straightforward. Android’s Java-based SDK allows some code reuse from Windows and Linux, while iOS development requires Swift and Mac hardware. Key considerations include responsive design, connectivity challenges, and leveraging native features like GPS, camera, and push notifications. |
| **Development Tools** | Objective-C, Swift (primarily for macOS/iOS development). Also supports Python, Java, and C/C++ as programming languages. Xcode has a cost of $99/year per developer which is required for iOS/macOS development.  May need expensive hardware and proprietary ecosystem for development. Secure due to Unix-based architecture, but software availability is limited. | Linux development may use languages such as C/C++, Java, Python, Rust, Go. Ideal for backend and system-level development. Highly secure and stable, commonly used for web hosting and cybersecurity.  IDE can include Eclipse, PyCharm, Visual Studio Code, and many other free/open-source options.  Cloud & DevOps:  Cost is free and open-source, making it the most cost-effective development environment. | Visual Studio ranges from $45 – $250 USD per user, depending on features, per year. Languages include C#, .NET, VB.NET, PowerShell scripting.  IDE used is Microsoft Visual Studio ($45–$250/year), widely used for enterprise applications.  Cross-Platform support is limited compared to Mac/Linux but supports some frameworks like Flutter.  The leading platform for game development with strong support for Unity, Unreal Engine, and DirectX.  More security vulnerabilities than Mac/Linux but benefits from frequent patches and extensive enterprise support.  Licensing costs can add up, but broad hardware compatibility makes it flexible. | Android SDK is Java based and the most widely used Android IDE is Android Studio which is developed by Google as the official development tool. Android Studio is free to download.  iOS uses Objective-C & Swift with Xcode ($99/year per developer). macOS is required for iOS development. No OS restrictions for development, but a Mac is needed to build iOS apps.  Cross-Platform development tools include Flutter (Dart-based) and React Native (JavaScript-based) allow development for both iOS and Android, but iOS builds require a Mac.  Other tools like Xamarin and Ionic offer additional cross-platform development options. |

## 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**

To help *Draw It or Lose It* grow and run on different devices, the best choice is a cloud-based server like Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). These platforms make it easy to expand, handle more users, and keep the game running smoothly.

1. **Operating Systems Architectures**

A Linux-based system is recommended because it is stable, secure, and cost-effective. Linux supports containerization tools like Docker and Kubernetes, which help in running and managing the game on multiple devices easily. This also makes updates and scaling more efficient.

1. **Storage Management**

To store game assets and player data effectively, we recommend using cloud storage services such as Amazon S3, Google Cloud Storage, or Azure Blob Storage. These offer flexible storage that grows as needed, backup and recovery to prevent data loss, and fast access to files from anywhere in the world. A relational database like PostgreSQL or MySQL should be used to store player progress and game settings in an organized way.

1. **Memory Management**

To ensure smooth gameplay, the recommended platform should use caching tools like Redis or Memcached to store frequently used data for quick access. Additionally, load balancing to share work between multiple servers, preventing slowdowns. Finally, the platform should use garbage collection and memory pooling to keep memory usage efficient and avoid crashes.

1. **Distributed Systems and Networks**

Since the game needs to work across multiple devices and platforms, we suggest using a microservices architecture where different parts of the game run separately. This makes the game more scalable, meaning it can handle more players without slowing down, easier to update, since changes to one part won’t break the entire game. And finally, more reliable as one issue won’t cause the whole game to crash.

For network communication APIs (REST or GraphQL) will allow the game to send and receive data efficiently, a Content Delivery Network (CDN) (such as Cloudflare or AWS CloudFront) will speed up loading times for images and game files, and message queues (RabbitMQ or Kafka) will help manage real-time updates and game events.

1. **Security**

Security is essential for protecting user information and preventing unauthorized access. The recommended security measures include:

**Data Encryption:** Protecting data with TLS/SSL encryption while it’s being sent and AES-256 encryption when stored.

**User Authentication:** Using OAuth 2.0 or JSON Web Tokens (JWT) to keep user accounts secure.

**Regular Security Checks:** Performing security scans and updates to prevent hacking.

**DDoS Protection:** Using tools like AWS Shield or Cloudflare to prevent cyberattacks.

**Access Controls:** Setting up Role-Based Access Control (RBAC) to limit user permissions and prevent unauthorized access.