

<Name-of-Software-Application>

# **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 | 10/15/25 | Aisha Anderson | Analyzing characteristics and various techniques, Updating Recommendation section |

**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 wants to expand their Android-only game, Draw It or Lose It, into a web-based version that works across multiple platforms. The game involves teams trying to guess puzzles based on images shown during timed rounds. To support this transition, Creative Technology Solutions (CTS) will help guide the setup and development process.

My plan is to build a system that allows multiple teams and players to join, while making sure each game and team name is unique. I’ll also make sure only one active game runs at a time by using unique IDs for each game, team, and player. This setup will help keep everything organized and prevent issues with duplicate sessions. The design will be flexible so it can grow with future updates and support more users across different devices.

## 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)

When developing Draw, It or Lose It as a web-based game, I need to make sure it works well across different platforms like desktops, tablets, and mobile browsers. The game must be responsive and run smoothly no matter what device someone is using. Since the game includes timed rounds and live image rendering, I’ll need to use tools that support real-time communication between the server and players like WebSockets or similar technologies. Another important part is making sure only one active game instance exists in memory at a time. To keep everything organized and avoid duplication, I’ll assign unique IDs to each game, team, and player.

I also need to make sure that game and team names are unique, so players don’t accidentally choose names that are already taken. This means building a system that checks name availability during setup. As more users join, the game must be able to handle the extra traffic without slowing down, so I’ll design it to be scalable and well-organized. Security is also a top priority; I’ll make sure user data and game sessions are protected with secure login and validation processes. These constraints will shape how I build the game’s structure, database, and user interface to make sure it’s reliable, safe, and easy for everyone to use.

## [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 for The Gaming Room shows how the different parts of the game system are connected. At the center is the Entity class, which acts like a blueprint for other classes such as Game, Team, and Player. This class includes shared features like an ID and a name, that helps keep things steady and decreases repeated code. Each of these classes inherits from Entity, which means they automatically get those shared features and can also add their own specific details. For example, a Game holds a list of Teams, and each Team holds a list of Players, which matches the requirement that games have multiple teams and teams have multiple players.

The GameService class is responsible for managing all the games. It keeps track of the list of games and includes methods to add, find, or remove games. It also uses a unique ID system to make sure each game, team, and player is easy to identify and doesn’t get mixed up. The ProgramDriver class is likely the starting point of the application, and SingletonTester helps make sure only one game instance is running at a time, which is important for keeping the system organized and efficient.

This diagram shows several object-oriented programming principles. Inheritance is used to share common features through the Entity class. Encapsulation keeps data secure by using private variables and public methods to access them. Abstraction helps simplify the design by focusing on shared traits and hiding unnecessary details. Composition is shown through how games contain teams and teams contain players. Finally, the Singleton pattern is used to make sure only one game instance exists in memory, which supports the client’s requirement for managing game sessions properly. All of these principles help make the software more organized, reusable, and easier to maintain.

**"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** | macOS is not used for hosting large-scale web applications. It supports Apache and Nginx; deployment is limited due to hardware constraints and lack of enterprise-grade server tools. Licensing costs are high since macOS requires Apple hardware. It’s more suitable for development and testing than production hosting. | Linux is the most popular platform for web hosting due to its scalability, reliability, and open-source nature. It supports Apache, Nginx, Node.js, and Docker, making it ideal for hosting the game server. There are no licensing fees, which keeps costs low for The Gaming Room. | Windows Server supports IIS and other web technologies, making it suitable for enterprise hosting. It integrates well with .NET applications. But licensing costs can be major depending on the number of users and server instances. | Mobile devices are not used for hosting web applications. They connect to the server via APIs or browsers. Hosting is handled by accepted platforms like Linux or Windows. |
| **Client Side** | Developing Mac desktops means to ensure compatibility with Safari, Chrome, and Firefox. Responsive design using HTML/JavaScript works well. Development costs are reasonable, but testing requires macOS hardware. Knowledge of Apple’s ecosystem may be needed. | Linux users normally access the game via Chrome or Firefox. Web standards ensure compatibility. Development costs are low, and testing can be done on virtual machines. Linux is developer-friendly and supports open-source tools. | Windows users access the game through Chrome, Edge, or Firefox. Responsive design ensures compatibility. Development costs are moderate, and Windows is widely used, making it a priority platform for testing. | Supporting iOS and Android requires open design and cross-browser testing. Native apps require Swift (iOS) and Java (Android), but using frameworks like React Native or Flutter can reduce time and cost. Knowledge in mobile UI/UX is important. |
| **Development Tools** | Development on Mac uses tools like Xcode (for iOS), VS Code, and Git. Languages include Swift, JavaScript, and HTML/CSS. Xcode is free, but Apple hardware is required. Teams may need macOS systems for iOS testing and deployment. | Linux supports a large range of free tools like Eclipse, VS Code, Git, and Docker. Languages include JavaScript, Python, Java, and PHP. It’s ideal for backend development and web deployment. One unified team can work across platforms. | Windows development uses Visual Studio, VS Code, and Git. Languages include C#, JavaScript, and HTML/CSS. Visual Studio Community is free, but Enterprise versions have licensing costs. Windows is great for .NET-based development. | Mobile development uses Android Studio (free) and Xcode (free but requires Apple hardware). Languages include Kotlin, Swift, Dart, and JavaScript. Hybrid frameworks like Flutter or React Native allow one team to support both platforms efficiently. |

## 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 work on desktop systems and mobile browsers, I recommend using a Linux-based server like Ubuntu Server. I chose Linux because it’s free to use, easy to customize, and works well in both development and live environments. It also connects smoothly with cloud services like AWS, Azure, and Google Cloud, which makes it a good choice for running multiplayer games and sharing data between platforms. Linux has strong security tools and can be adjusted to fit the game’s needs, so it’s a solid option for handling more users and making the game available across different devices.
2. **Operating Systems Architectures**: I chose Linux as the operating system for Draw It or Lose It because of its strong architecture and flexibility. Linux uses a monolithic kernel, which means it combines important services like memory management, process control, and device drivers into one efficient system. This setup helps the server run faster and more smoothly. I also like the fact that Linux is modular, so I can add or remove parts of the system depending on what the game needs. It supports multitasking and multiple users at the same time, which allows me to run many game sessions without issues. Since Linux follows POSIX standards, I can also move the game to other similar systems more easily. This architecture helps with fast image rendering, tracking game sessions, and quick responses, all of which are important for giving players a smooth and enjoyable experience.
3. **Storage Management**: I would use cloud-based solutions like Amazon S3 or Google Cloud Storage to host the game’s high-quality image assets. These services are secure, scalable, and fast, and they work well with content delivery networks to ensure quick loading times for players around the world. I would manage structured data like player accounts, scores, and game history using a relational database such as MySQL. If the game includes features like chat or flexible user-generated content, I would consider using a NoSQL database like MongoDB. To protect this data, I would set up regular backups, control access permissions, and define data retention policies. These steps help me ensure that the game remains scalable, secure, and responsive as the user base grows.
4. **Memory Management**: I would like to apply several techniques to keep the game running smoothly. Since each image is about 8MB and the game uses many of them quickly, I wouldn’t load all images into memory at once. Instead, I would use lazy loading to only bring in the images needed for the current round. I’d also use memory pooling to reuse memory blocks and reduce overhead. To optimize performance further, I’d use image formats like WebP or AVIF, which maintain high quality while using less memory. On web browsers, I’d rely on caching to help repeat users load images faster. On the server side, I’d use Redis to store active game sessions in memory, which is much faster than using disk-based storage. I’d also implement monitoring tools to track memory usage and clean up unused data, especially when many games are running at the same time.
5. **Distributed Systems and Networks**: To make Draw It or Lose It work smoothly on different platforms and devices, I would use a setup called a distributed system. This means the main server would handle the game’s rules and actions, while players on phones, computers, or browsers would connect to it using tools like APIs or WebSockets. I would use load balancing to spread out traffic, so no single server gets overwhelmed, and I’d use cloud auto-scaling to add more power when lots of people are playing. I’d also use CDNs to send images quickly to players no matter where they are. To keep things consistent, I’d add retry options for dropped connections, use backup servers in case something goes wrong, and make sure data moves quickly and smoothly. This setup would help me keep the game running well and give players a steady experience across all platforms.
6. **Security**: Security is very important when dealing with user data across different platforms, so I would take several steps to keep it safe. I would use encryption to protect data while it’s being sent (TLS/SSL) and while it’s stored (AES). To make sure users log in safely, I would use secure login systems like OAuth 2.0 or JWT. On the server, I would use Linux tools like SELinux or AppArmor to control who can access important parts of the system. On mobile devices, I would store sensitive information using built-in secure storage like Keychain for iOS and Keystore for Android. I would also follow privacy laws like GDPR and CCPA to protect user rights. Finally, I would set up logs and monitoring tools to keep track of activity and catch any problems early. These steps would help me keep user data safe and build trust with players.