

Draw it or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | <11/18/24> | Ben Schmidt | Executive Summary, Design Constraints and Domain Model updated. |
| 1.1 | <12/4/24> | Ben Schmidt | Updated the Evaluation section. |
| 1.2 | <12/16/24> | Ben Schmidt | Completed recommendations 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 is developing Draw It or Lose It, a web-based game where teams guess puzzles as drawings are revealed. The game needs to support multiple teams and players, with unique names for games and teams to avoid confusion. To ensure only one game runs at a time, we’ll use the singleton design pattern. The system will also validate names when adding games, teams, or players using an efficient search method. The design will feature a shared Entity class for common attributes like IDs and names, with specific classes for games, teams, and players built on top of it. Timed gameplay mechanics will manage rounds, ensuring smooth progress and fair play. The solution is organized, scalable, and ready for future improvements, meeting the needs of the client while keeping the development process efficient.

## 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)

Developing Draw It or Lose It in a web-based distributed environment comes with several design challenges. First, scalability is important to support many players and teams while maintaining performance. This requires managing resources well, optimizing database queries, and improving network communication. Second, ensuring data consistency and synchronization is crucial since players interact in real-time, requiring a strong backend and synchronization protocols. Third, high availability is needed to handle server downtimes or network issues, so redundant server setups or cloud solutions are necessary. Fourth, security is a concern, particularly for validating unique names and protecting user data, requiring secure communication and authentication. Lastly, modularity and maintainability are key for future updates, requiring an object-oriented design with clear separation of concerns and reusable code. These constraints drive the development process, focusing on strong backend systems, efficient coding, and thorough testing for a smooth user experience.

## [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 represents a gaming system with classes and relationships that demonstrate key object-oriented principles. The Entity class serves as a base for Game, Team, and Player, encapsulating shared attributes like id and name through inheritance. The GameService class acts as a singleton, managing unique IDs and providing methods for adding and retrieving games, teams, and players, ensuring efficient resource use. Game contains a list of Team objects, and Team contains a list of Player objects, showcasing composition and real-world relationships. Encapsulation protects data integrity by limiting access to private attributes, while polymorphism is evident in the toString method, which is overridden across multiple classes for custom behavior. The design prioritizes scalability, maintainability, and abstraction, with methods like addGame, addTeam, and addPlayer simplifying object creation and organization, making it well-suited to meet software requirements efficiently.

**"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 can host a web-based application, but it's not as widely used for high-scale deployments as windows. Licensing: macOS is free for personal and development use. For commercial use, particularly for hosting, there may be additional licensing for server software (e.g., Apple Xserve, if used). Xcode and macOS development are free, but enterprise-level services like Apple Business Support could incur fees. Cost Estimate: Xcode and macOS: free. Apple Business Support or Server software (Apple Xserve): can range from $500 to $5,000 depending on scale. | Linux is highly customizable, open-source, and free. Most Linux distributions, such as Ubuntu and CentOS, are free to use for web hosting. Licensing: Linux itself is free, but enterprise Linux distributions (Red Hat Enterprise Linux, SUSE Linux Enterprise) require licensing. Cost Estimate: Red Hat: $349 to $2,000 per year per server depending on support level. SUSE: starts around $799 per server per year. | Windows Server is commonly used for web hosting and enterprise applications. Licensing: Windows Server requires a paid license for both the operating system and CALs (Client Access Licenses). Cost Estimate: Windows Server 2022 Standard Edition starts around $972 (per license), with additional CALs costing $30 to $100 each. Total cost could range from $1,000 to $10,000 depending on scale and required features. | Mobile devices themselves do not host web applications. Instead, cloud hosting or dedicated servers are recommended. Licensing: The cost for using cloud services is based on the provider (e.g., AWS, Google Cloud, Azure). Mobile devices for app development have no direct licensing fees, but hosting services like AWS or Azure would require payment. Cost Estimate: Cloud hosting costs can vary widely, starting at around $0.05 to $0.20 per hour for basic computer resources (AWS EC2), with additional fees for storage, data transfer, etc. |
| **Client Side** | macOS offers great support for web browsers (Safari, Chrome, Firefox), but the app must be tested and optimized for multiple browsers. Licensing: macOS itself is free to use. No additional licensing costs for browser-based apps. If creating a native macOS app, Xcode is free, but some third-party tools (e.g., for game development) may have licensing costs. Cost Estimate: Xcode and macOS development tools are free. Third-party tools like Unity may cost around $399/year for the Pro version. | Linux is highly open-source and free for developing and testing web applications. Licensing: No direct costs for using Linux distributions, and development tools like VS Code are free. If using premium tools (like JetBrains IDEs or certain proprietary tools), licenses would be required. CostEstimate: JetBrains tools: around $49 to $249 per year depending on the product (e.g., WebStorm). | Windows is commonly used for web development with browsers like Edge, Chrome, and Firefox. Licensing: No direct cost for using Windows for web development. However, if creating native Windows apps, development tools like Visual Studio require licensing. Cost Estimate: Visual Studio Community Edition is free, but Visual Studio Professional costs $45/month or $1,199 for the first year. | Mobile development requires native apps for iOS and Android or cross-platform development tools like React Native. Licensing: For iOS, a Mac is required for development, and an Apple Developer Program membership is necessary for app distribution. Android development tools are free, but mobile app stores charge for publishing. CostEstimate: Apple Developer Program: $99/year. Google Play Store: $25 one-time fee for app registration. Cross-platform tools like React Native or Flutter are free but may require third-party tools with associated fees. |
| **Development Tools** | Xcode (Swift, Objective-C) for macOS apps; HTML5, CSS3, JavaScript for web development. Game development may require Unity or Unreal Engine. Licensing: Xcode is free, but game engines like Unity may require a Pro license. CostEstimate: Unity Pro license: $399/year. Unreal Engine: free for small projects (fees start once revenue exceeds $1M/year). | Linux supports a range of free tools like VS Code, Python, and JavaScript frameworks (React, Angular). Licensing: Most development tools are free, but some may require licenses for advanced features (e.g., JetBrains IDEs). Cost Estimate: JetBrains WebStorm: $49/year. | Windows supports tools like Visual Studio (C#, C++, .NET) and game engines (Unity, Unreal). Licensing: Visual Studio offers free and paid versions. Other tools may have licensing costs. Cost Estimate: Visual Studio Professional: $45/month or $1,199/year. Unity Pro: $399/year. Unreal Engine: free until revenue exceeds $1M/year. | Mobile app development requires platforms like Android Studio (Java/Kotlin) and Xcode (Swift), or cross-platform tools like React Native or Flutter. **Licensing**: Xcode and Android Studio are free, but certain third-party tools may require a license. CostEstimate: Apple Developer Program: $99/year. Google Play Store registration: $25 one-time fee. |

## 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:**

I recommend deploying Draw It or Lose It on a Linux-based server platform. Linux is ideal for hosting web-based applications due to its scalability, reliability, and low licensing costs. It can handle high-traffic loads and supports key web technologies such as Apache, NGINX, MySQL, and PHP. Additionally, Linux has strong compatibility with cloud services (AWS, Azure, and Google Cloud), which simplifies scaling as the game expands.

2. **Operating Systems Architectures:**

Linux follows a modular, open-source architecture that allows customization based on the server's specific requirements. It leverages the POSIX-compliant architecture, supporting multi-user environments, robust threading, and process management. Distributions like Ubuntu Server or CentOS are widely adopted for web hosting and offer long-term support, ensuring stability for enterprise-level deployments. Linux also supports virtualization, which allows for efficient resource utilization across servers.

3. **Storage Management:**

For storage management, I recommend implementing a relational database system (RDBMS) such as MySQL or PostgreSQL for structured game data, including user profiles, game states, and team scores. These databases are optimized for web-based applications, providing strong performance, scalability, and data integrity. In addition, Amazon S3 or similar cloud-based storage solutions can be integrated to store static assets like stock drawings, ensuring reliable and accessible storage.

4. **Memory Management:**

Linux employs advanced memory management techniques to ensure efficient resource allocation. It uses virtual memory to manage large processes and incorporates paging and swapping to prevent memory overload. The Linux kernel's dynamic memory allocation ensures optimal use of physical RAM, while caching and buffer mechanisms enhance application performance. For Draw It or Lose It, in-memory caching solutions like Redis or Memcached can further improve speed by reducing database queries and enabling real-time data retrieval.

5. **Distributed Systems and Networks:**

To enable seamless communication across platforms, I recommend deploying Draw It or Lose It using a distributed client-server architecture. The web application will serve as the back-end system hosted on the Linux server, communicating with client devices (Windows, macOS, Linux, iOS, and Android) via HTTP/HTTPS protocols. Load balancing tools like HAProxy or cloud-based solutions can distribute traffic efficiently across servers, ensuring reliability during high demand.

For networking, using RESTful APIs will allow client platforms to interact with the server securely and consistently. In case of connectivity issues or outages, incorporating redundancy techniques such as database replication and failover servers will maintain system availability and performance.

6. **Security:**

Security is a top priority to protect user data and ensure trust. The Linux operating platform offers strong security measures, including:

Data Encryption: Encrypting user data in transit using SSL/TLS and at rest with tools like AES-256.

Firewalls: Configuring firewalls to prevent unauthorized access.

Authentication: Implementing multi-factor authentication for user accounts and OAuth 2.0 for secure logins.

Secure APIs: Protecting endpoints with proper authentication and rate limiting.

Regular Patching: Regular updates and patches for the Linux system to mitigate vulnerabilities.

Additionally, compliance with data privacy regulations like GDPR and CCPA will ensure user information is handled securely across all platforms.