

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 08/02/2025 | Brandon Burks | Evaluation of server-side, client-side, and development tools of various OS. |

**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 has successfully launched the Draw it or Lose it game as an Android Application. Attempting to reach a broader audience, CTS proposed a web-based version of the game that preserves core gameplay, such as, four timed rounds of rendered clues and team guessing, while also adding cross-platform access. Our design will leverage a modular and service-oriented architecture, with a singleton to manage a single active game instance in memory, while ensuring unique identifiers for games, teams, and players. By basing all domain objects on a common entity superclass and exposing RESTful APIs for game, team, and player management, we can enable consistent business logic, Streamlined name-uniqueness check, and simplified integration with front and back-end components.

## 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)

* Stateless Web Protocols: HTTP is inherently stateless, therefore the application must manage session state, such as active game, current round, team turns, on the server or by using secure cookies.
* Browser Compatibility: The Client must have ability to render consistently across modern browsers such as Chrome, Firefox, Safari, Edge, and devices such as, desktop, tablets, and mobile, requiring a responsive UI Framework and cross-browser testing.
* Concurrency and Scalability: Although only one game instance exists per server process, multiple games can run concurrently on a different server node. The Unique identifier generation and name-uniqueness checks must handle the concurrent requests safely.
* Performance: Real-time clue rendering and timer accuracy will demand low-latency communication and efficient data models.
* Security: All endpoints must enforce authentication to prevent unauthorized game manipulation. Unique game, team, and player names must be validated on the server-side to avoid injection attacks and naming conflicts.

## [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 centers on an abstract Entity superclass, attributes of id: long, name: String, from which Game, Team, and Player inherit, demonstrating inheritance and polymorphism.

* GameService(Singleton): Manages the creation and lookup of Game, Team, and Player instances. Private counters like nextGameId, nextTeamId, and nextPlayerId ensures unique identifiers, and a private constructor, getinsance() enforces the singleton patter, guaranteeing one service, and one active game context in memory.
* Game: Contains a list of Team objects (multiplicity 0..\*). Methods addTeam(name: String) creates and associates new Team instances under the game instance, enforcing composition and encapsulation of game logic.
* Team: Contains a list of Player objects. The addPlayer(name: String) method encapsulates the player creation.
* Player: Leaves association with Team (Teams may exist in the system), however, in practice, each player is created within a single team context.

**"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** | Unix based, supporting Docker and server hosting for developing and testing. This approach is not scalable or cost effective for production due to the high hardware cost as well as limited enterprise-grade server options. The licensing is expensive and strongly dependent on hardware. | The industry standard for web-based hosting. Linux is open-open source and free to use and is widely supported by most cloud providers. This option requires a strong knowledge of command-line. With no licensing fees, this is ideal for large-scale deployments. | Windows server supports hosting with .NET and IIS integration but has a higher licensing and resource cost compared to other options. This is an ideal option for .NET heavy stacks, but less cost-efficient. | Mobile platforms are not feasible for server side hosting due to the hardware constraints like CPU and memory, and should only be used for client access to the server-based application |
| **Client Side** | Supports HTML5 web apps on modern browsers such as Safari, Chrome, Firefox, however developing and testing requires Mac hardware. Additional provisioning efforts are needed. | Has broad and flexible browser support, but the differences across Linux distributions may add some complexity to QAand UI consistency testing. | Offers a wide browser and plugin compatibility (Chrome, Firefox, Edge), however development time can be increased with the need of legacy support (internet explorer). | Mobile clients require a responsive or hybrid web app development to support Apple IOS and Android. This adds complexity in the UI design, testing, and handling of SDK due to device differences. Tools such as React Native or Flutter can help ease the development. |
| **Development Tools** | Tools: Xcode (IOS), VSCode, WebStorm, Homebrew, Docker. These are used for IOS building as well as general web development. Xcode requires macOS. Some of these tools are free but only on Apple hardware. | Tools: VSCode, Eclipse, Vim/Emacs, Docker, Kubernetes, apt/yum. These are idea for backend and server work. Majority of the tools are open-source and free, enabling low-cost and scalable team setups. | Tools: Visual Studio (They have a paid version and a free community version), VSCode, IIS, WSL for Linux-like dev environments. This option is good for .NET or cross-platform dev. There may be some licensing cost for visual studio pro. | Tools: Android Stufio (free), Xcode macOS only, React Native, Flutter. Multiple SDKs are needed. Development teams must be familiar with cross-platform mobile development. |

## 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**: The recommended operating platform for The Gaming Room is Linux-based servers, more specifically a distribution such as Ubuntu or Red Hat Enterprise Linux. Linux has set the industry standard for web hosting due to its scalability, stability, and broad support it receives from all major cloud providers such as AWS, Azure, and Google Cloud. There are no licensing fees, supports containerization, like Docker and Kubernetes, and is also cost-effective for production environments. Windows servers are well-suited for .NET-heavy apps, but linux provides a more flexible, light weight, and cloud-native environment that aligns with client expansion goals.
2. **Operating Systems Architectures**: The recommended platform architecture should follow a 64-bit Linux Operating system, which takes advantage of multi-core CPUs for concurrent connections and memory allocation that is more efficient. A client-server architecture should be employed with the back-end hosted on Linux servers and front-end clients (browsers or mobile devices) connecting via RESTful APIs. This architecture will support modular scaling: horizontal scaling for addition concurrent users. And a vertical scaling for when more CPU or memory resources are needed.
3. **Storage Management**: A relational database management system like PostgreSQL or MySQL is recommended for handling the structured game data like users, teams, and sessions. These systems will provide robust indexing, ACID compliance, and transactional safety. For scalability and redundancy, this database can be hosted in the cloud-based service with addition of automated back ups and replications. Additionally, using a service such as Amazon S3 will allow for static assets like images and clue libraries.
4. **Memory Management**: Linux servers can also efficiently manage memory through virtual memory and paging, allowing processes to use the memory dynamically. For Draw it or Lose it, real-time performance is critical. Using in-memory caching systems like Redis or Memcached will allow frequently accessed data to be quickly retrieved, reducing latency. Garbage collection in Java or other runtime environments will prevent memory leaks while ensuring stability during long-running sessions.
5. **Distributed Systems and Networks**: Cross-platform communication can be achieved by using a service-oriented architecture supported by RESTful APIs and WebSockets. The application should be deployed in a cloud environment with load balancers, which will ensure availability and automatic failover in case of an outage. Containerization like Docker + Kubernetes can support multiple concurrent game servers, each running isolated game instance. This type of setup minimizes the downtime and improves the resilience against hardware failures while also maintaining low latency for real-time clue rendering and guessing.
6. **Security**: Security is essential to protect the user’s information across all platforms. Key measures that can be included are:

* Authentication and Authorization: Implement JWT or OAuth 2.0 for a more secure login and access control.
* Data Protection: All communication between the client and server should be used for TLS/SSL encryption.
* Server-Side Validation: Input sanitization and validation will protect against injection attacks and naming conflicts.
* Database security: Encrypt sensitive fields at rest and enforce role-based access control within database operations.
* Monitoring and Logging: Intrusion detection, activity logging, and using audit trails will ensure that breaches can be detected and addressed quickly.

By Leveraging the Linux servers paired with cloud-based deployment, relational databases, in-memory caching, and secure distributed architecture, The Draw it or Lose it application can easily scale to support multi-platform access while also maintaining performance, reliability, and user data protection.