

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 | 04/20/2025 | Erin Stadler | Revision 2- final revision |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is excited to announce plans for expanding its popular Android game, "Draw It or Lose It," into a versatile web-based platform that caters to a variety of environments. Our objective is to develop a scalable and efficient game application that ensures seamless user experience across all devices.

To achieve this, we will utilize effective software design patterns that align with our client's requirements. Key features will include the capability for unique team names and game names, robust management of multiple players per team, and the assurance of a single instance of the game maintained in memory. By implementing these strategies, we aim to create an application that is not only high-performing but also easily maintainable and scalable for future growth.

## Requirements

*The application must fulfill the following business and technical requirements:*

* *Support multiple teams, each consisting of several players.*
* *Ensure unique names for games and teams to prevent duplication.*
* *Maintain only one instance of the game in memory at any given time.*
* *Assign unique identifiers to each game, team, and player instance.*
* *Utilize appropriate software design patterns to ensure maintainability and scalability.*

## [Design Constraints](#_2et92p0)

Developing the game in a web-based distributed environment presents several constraints:

* **Scalability:** The game must efficiently handle multiple concurrent users, which requires optimized database interactions and backend processes.
* **Consistency:** To ensure unique game and team names, a robust data validation and storage mechanism is necessary.
* **Memory Management:** A single instance of the game should be maintained in memory using a singleton pattern to ensure efficient resource allocation.
* **Platform Compatibility:** The game must function seamlessly across various operating systems and browsers.
* **Security:** User data must be protected through secure authentication methods and data encryption.

These constraints significantly influence software design decisions, including database selection, programming frameworks, and hosting solutions.

## [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 provided illustrates the relationships between the core entities in the system:

* **Entity Class:** This is a base class that contains common attributes, such as id and name.
* **Game Class:** This class inherits from Entity and represents an instance of a game.
* **Team Class:** This class, derived from Entity, allows for the inclusion of multiple players per team.
* **Player Class:** This class also inherits from Entity and represents individual participants in the game.
* **GameService Class:** This class implements the singleton pattern to ensure that only one instance of the game exists.

The object-oriented principles applied in this design include:

* **Encapsulation:** Each class has private attributes with public getter and setter methods.
* **Inheritance:** This reduces redundancy by allowing the Game, Team, and Player classes to inherit common properties from the Entity class.
* **Singleton Pattern:** This pattern is used to maintain a single instance of the GameService.
* **Iterator Pattern:** This pattern enables easy traversal of collections, facilitating the addition and retrieval of unique games and teams.

**"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)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Development Requirements | Mac | Linux | Windows | Mobile Devices (iOS & Android) |
| Server Side | The software supports server-based deployment on macOS Server; however, it has limited utility for enterprise use and comes with higher licensing costs. | The most popular server operating system for web applications is free, open-source, highly scalable, and secure. | This software is compatible with IIS and Windows Server. However, licensing costs can be high, and it often requires more system resources. | This product is not intended for server hosting and is meant for client-side use only. |
| Licensing Costs (Server) | A paid license is required for macOS Server. | Free and open-source operating systems, such as Ubuntu and CentOS. | Paid licenses are required for Windows Server and Client Access Licenses (CALs). | N/A |
| Client Side | Safari support is beneficial for web development, though it may require minor optimizations. | Access is available through browsers like Firefox and Chrome, though some adjustments may be necessary. | The application is compatible with all major web browsers. Since most users operate on Windows, it ensures wide compatibility. | Must support both Android and iOS platforms. Requires a responsive design and thorough testing across various device types and screen resolutions. |
| Development Tools | Xcode (for iOS), IntelliJ, JavaScript frameworks (React). | Eclipse, IntelliJ, VS Code, JavaScript, Node.js. | Visual Studio, .NET, IntelliJ, JavaScript frameworks. | Android Studio (Java/Kotlin), Xcode (Swift), React Native, Flutter for cross-platform. |
| Development Cost & Time | iOS development requires macOS, which can lead to higher hardware costs. Additionally, if you're using web technologies, the time investment may be moderate. | Cost-effective solutions and faster iteration cycles, along with strong support from a broad development community. | Requires licenses for tools such as Visual Studio Pro. Development time may increase due to platform-specific issues. | Testing on various devices is necessary. Cross-platform tools like React Native and Flutter can help reduce effort and costs. |
| Technical Impact | Specialized team members may be required for iOS and macOS development. | It is easy to find and hire talent with experience in Linux servers, which helps facilitate robust CI/CD pipelines. | The complexity of the platform may lead to an increased demand for team resources. Additionally, there will be improved access to support for developers working with Windows systems. | A mobile development or cross-platform team is needed. The performance of the app can vary depending on the framework used, whether it is native or hybrid. |
| Browser Compatibility | Safari, Chrome, and Firefox offer high compatibility. | Compatible with Firefox, Chrome, and other browsers. | Chrome, Edge, and Firefox are all fully compatible. | HTML5 must be responsive and designed with a mobile-first approach. Chrome and Safari are the primary browsers for Android and iOS. |

## Recommendations Operating Platform

## Linux, particularly distributions like Ubuntu Server and CentOS, is highly recommended for server-side hosting. This platform is known for its superior performance, security, scalability, and cost-effectiveness, largely due to its open-source nature. Furthermore, it integrates seamlessly with cloud environments such as AWS, GCP, and Azure. On the client side, employing a web-based front end ensures compatibility across various devices, including Windows, macOS, Linux, iOS, and Android, by utilizing responsive design alongside modern web frameworks like React or Angular.

## Operating Systems Architectures

## A microservices architecture is optimal for this project, as it allows each service, like authentication, game logic, team management, and leaderboard handling, to function independently while communicating through APIs. This architecture supports scalability, fault tolerance, and independent deployment. Linux enhances this setup through effective containerization with tools like Docker and orchestration using Kubernetes, facilitating easier deployment and maintenance.

## Storage Management

## For data management, a hybrid approach using both PostgreSQL (relational) and MongoDB (NoSQL) is advisable:

## - PostgreSQL is recommended for managing structured data, including user accounts, teams, and game metadata.

## - MongoDB, on the other hand, is suited for handling semi-structured or session-based data, such as real-time game states and player interactions.

## Employing cloud-managed services like AWS RDS and MongoDB Atlas can further enhance scalability, ensure automated backups, and provide geographic redundancy for improved availability.

## Memory Management

## Java's built-in automatic garbage collection (GC) and memory optimization capabilities facilitate the efficient use of system resources. The Singleton design pattern within the GameService ensures that only one instance of the game remains in memory, thereby minimizing overhead. Additionally, tuning the Java Virtual Machine (JVM) in terms of heap size and GC algorithms, coupled with profiling tools like VisualVM, can optimize performance as the user base expands.

## Distributed Systems and Networks

## To enable effective communication across platforms, the following strategies are recommended:

## - Implement \*\*RESTful APIs\*\* for standard operations such as login, team creation, and game initiation.

## - Utilize \*\*WebSockets\*\* for real-time connections between players and the server, accommodating features like drawing updates and timer countdowns.

## - Deploy services on a cloud provider equipped with load balancing and fallback mechanisms to ensure reliability during outages.

## - Integrate API gateways and service registries for dynamic management of microservices.

## These practices enhance inter-service connectivity, reduce latency, and bolster resilience during partial system failures.

## Security

## To safeguard user data across various platforms, it is vital to:

## - Adopt OAuth 2.0 and OpenID Connect for secure authentication processes.

## - Ensure all client-server and inter-service communications are encrypted using HTTPS/TLS.

## - Implement Role-Based Access Control (RBAC) to secure administrative functionalities.

## - Store sensitive information with field-level encryption and utilize hashed passwords, employing algorithms like bcrypt.

## - Enable logging and intrusion detection with tools such as Fail2Ban, audit logs, and SIEM systems.

## Incorporating regular security updates and penetration testing into the DevOps pipeline will further strengthen the overall security posture of the system.e.

1. Operating Platform: For server-side hosting, Linux is the preferred operating system due to its stability, security, and cost-effectiveness. On the client side, the game should be fully compatible with all major operating systems and mobile devices, accessed through a web-based interface.
2. Operating Systems Architectures: The design of the game should incorporate microservices architecture. This approach allows for the independent management of components such as game logic, authentication, and data storage, thereby enhancing both maintainability and scalability of the application.
3. Storage Management: It is advisable to utilize a cloud-based relational database, such as PostgreSQL or MySQL, for structured data management. Additionally, NoSQL databases like MongoDB can be employed for the dynamic storage of game session data, accommodating flexible data requirements.
4. Memory Management: The implementation of the singleton pattern within the GameService will ensure efficient memory management by maintaining a single instance of the game at any given time. In addition, Java’s garbage collection will assist in freeing up unused memory, optimizing the overall performance of the application.
5. Distributed Systems and Networks: The use of RESTful APIs will streamline communication between different platforms. To enhance real-time updates and facilitate team interactions during gameplay, WebSockets can also be employed effectively.
6. Security: For user authentication, implementing OAuth 2.0 is highly recommended. It is crucial to use HTTPS for data encryption, along with secure practices for database management to ensure data integrity. Furthermore, role-based access control (RBAC) should be in place to safeguard sensitive game operations.