

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <1/31/25> | <Wilson Walker> | <Brief description of changes in this revision> |

**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](#_heading=h.35nkun2)

The Gaming Room plans to expand its Android game, *Draw It or Lose It*, into a web-based version that supports multiple platforms. This game, similar to *Win, Lose, or Draw*, challenges teams to guess images selected from a stock drawing library. The goal is to transition the game to an online environment while preserving key features such as player and team management, puzzle-solving mechanics, and timed gameplay. The development approach will incorporate object-oriented programming, design patterns like Singleton, and efficient resource allocation to enhance scalability and cross-platform functionality.

## Requirements

**Business Requirements:**

1. **Cross-Platform Support:** The game must be accessible on various platforms, including Windows, Mac, Linux, and mobile devices. Expanding beyond Android will help grow the user base and broaden its reach.

2. **Team-Based Competition:** The game should enable multiple teams to compete against each other while maintaining the core mechanics of the original version.

3. **Independent Game Sessions:** The system must allow multiple game instances to run at the same time, ensuring that each team plays its own match without interference from others.

4. **Consistent Branding:** The visual design and user experience of the web-based version should align with the current Android app to maintain brand identity.

**Technical Requirements:**

1. **Scalability and Concurrency:** The system must efficiently manage numerous users, teams, and game sessions at once while preventing performance issues such as server overload.

2. **Security Measures:** Sensitive information, including usernames and passwords, must be protected to prevent unauthorized access.

3. **Session Tracking:** Each game session should be individually managed, allowing users to reconnect if they get disconnected.

4. **Optimized Performance:** The web version should provide smooth gameplay with quick load times and responsive interactions, regardless of the user's device.

5. **Data Persistence:** Player progress and game session data should be securely stored so users can revisit past sessions or review their game history.

## [Design Constraints](#_heading=h.1ksv4uv)

1. **Multi-Platform Compatibility:** The game must function smoothly across various operating systems, including Windows, Mac, Linux, iOS, and Android. Achieving this requires the use of web technologies that support cross-platform deployment.

2. **Concurrent Gameplay:** The system must handle multiple teams and players interacting in real time, which may present challenges in session management and game state synchronization.

3. **Scalability:** As the number of teams and players grows, the application must efficiently scale. This includes managing game sessions, preserving session data, and optimizing server resources to maintain performance.

4. **Data Security:** Safeguarding user information and preventing unauthorized access is essential. Secure communication between users and the server, along with encryption for sensitive data, will be required.

## [System Architecture View](#_heading=h.44sinio)

1. **User Interface (Presentation):**

- The game's front-end will be designed for web browsers and mobile devices, providing players with an interactive platform. This layer will manage user inputs, display game data, and facilitate essential gameplay features such as initiating matches, joining teams, and solving puzzles. It will be built using widely used web technologies like HTML5, CSS, and JavaScript.

2. **Game Logic (Application):**

- This layer handles the core game mechanics, overseeing player interactions, tracking game progress, managing teams, and rendering puzzles. It will use server-side technologies such as Java or Node.js to facilitate communication between the client and database, ensuring smooth operation and enforcing game rules.

3. **Data Management (Storage):**

- The database will store all game-related information, including player details, team progress, game states, and puzzle sets. A relational database or a NoSQL alternative will be utilized to ensure secure storage and efficient data retrieval. Maintaining data integrity is critical to prevent loss of progress and allow seamless reconnections for users.

4. **Networking and Real-Time Communication:**

- The client and server will communicate through a secure HTTP/S connection. Real-time game updates may be handled using WebSockets or server-sent events to ensure minimal delay in gameplay interactions, such as puzzle solving and team coordination. Synchronization between the game instances and server will maintain consistency across all platforms.

5. **Security and Access Control:**

- Security measures will be put in place to protect player data and restrict access to authorized users. Authentication and authorization mechanisms will ensure players only access their own sessions. Encryption protocols like HTTPS and secure storage methods will help safeguard sensitive information and maintain a protected gaming environment.

## [Domain Model](#_heading=h.2jxsxqh)

**- ProgramDriver:** Responsible for initializing and launching the primary application.

**- SingletonTester:** Used to verify the implementation of the Singleton pattern within the system.

**- Entity Class:** Functions as the foundational class, incorporating common attributes such as id and name, along with methods like getID(), getName(), and toString(). This class is extended by key components such as Game, Team, and Player, promoting code reusability and adhering to object-oriented principles like inheritance.

**- GameService Class:** Implements the Singleton pattern to oversee game-related operations, ensuring that only one instance of this class is created. It maintains a collection of games and provides methods to add, retrieve, and count them while also managing player and team identifiers.

**- Game, Team, and Player Classes:** Represent the essential components of the game, each handling its own data structures. These classes utilize object-oriented techniques such as composition to define relationships between objects. For instance, a Game object contains a list of Teams, and each Team holds a list of Players.

**"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](#_heading=h.z337ya)

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** | Known for its secure and stable environment, macOS is viable for hosting web applications but is rarely chosen for large-scale deployments. Licensing costs for Mac servers are high, and scalability options are limited. However, it offers seamless integration with development tools. | One of the most widely used server environments, Linux is valued for stability, scalability, and affordability. It supports open-source tools and is highly customizable. Strong community support makes it ideal for hosting large-scale web applications at minimal cost. | A dependable choice for enterprise environments, Windows Server supports various development frameworks. However, its licensing fees are higher compared to Linux. It is well-suited for .NET applications and integrates well with Microsoft technologies. | Not ideal for hosting server applications, but crucial for server-client interactions. Optimizing server-side interactions requires efficient REST API use, WebSockets, and proper bandwidth management to maintain performance across mobile networks. |
| **Client Side** | Provides a robust development environment with native support for Xcode and Swift. However, cross-platform compatibility is somewhat limited. Web applications must be tested thoroughly on other platforms to ensure proper functionality. | Linux supports multiple distributions and development environments, but client-side development can be challenging due to compatibility requirements with macOS and Windows. Despite this, it remains a powerful option for web-based applications. | The dominant platform for desktop clients, Windows supports popular browsers like Edge, Chrome, and Firefox. Applications must be optimized for varying hardware configurations. Development is generally straightforward, ensuring compatibility across different browsers. | Requires responsive design and optimization for multiple screen sizes and devices. iOS and Android have different development environments, necessitating thorough cross-platform testing. Using frameworks like React Native, Flutter, or Xamarin can streamline development. |
| **Development Tools** | Developers use Xcode, Swift, and other Apple-specific tools, though cross-platform tools like Visual Studio Code and Eclipse are available. High licensing costs for Mac hardware can be a drawback. | Linux supports open-source tools such as Eclipse, VS Code, and Vim. It allows extensive customization, making it a preferred choice for server-side development. Common languages include Node.js, Python, and Java. | Windows developers frequently use Visual Studio, a powerful integrated development environment for cross-platform applications. It supports multiple programming languages like C#, Java, JavaScript, and Python. However, enterprise versions require licensing fees. | Mobile development relies on platform-specific tools like Xcode for iOS and Android Studio for Android. Cross-platform frameworks like React Native, Flutter, and Xamarin provide unified codebases for both operating systems, reducing development time and effort. |

## 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 preferred operating system for *Draw It or Lose It* is Linux due to its stability, scalability, and cost-effectiveness, making it well-suited for hosting web applications. Its open-source nature helps minimize licensing costs, and it offers strong support for server-side frameworks like Node.js, Python, and Java. Given that the game involves multiple teams and simultaneous user interactions, Linux's efficient concurrency handling makes it an ideal choice. Additionally, it integrates seamlessly with cloud services, making it a strong option for scaling the application as the user base expands.
2. **Operating Systems Architectures**: A client-server model is recommended, where the client application, whether on desktop or mobile, handles the user interface and interactions, while the server manages game logic, database operations, and player communication. Linux servers will be used to host core game processes, databases, and web services, while the client-side will be accessible via web browsers or mobile devices. This architecture ensures security, scalability, and simplified maintenance across platforms.
3. **Storage Management**: A relational database like MySQL or PostgreSQL is suggested for managing stored data. These databases provide high data consistency, support for complex queries, and reliable transactional capabilities—key features for handling sessions, player profiles, and team information. To enhance performance, indexing and optimization strategies will be employed to efficiently process large datasets. For scalability, cloud-based storage solutions such as Amazon RDS or Google Cloud SQL can be used for backup, recovery, and seamless expansion.
4. **Memory Management**: To improve performance, memory management techniques like caching will be used to reduce server and database load. Tools such as Redis or Memcached can be utilized to temporarily store frequently accessed data, speeding up game response times. In addition, garbage collection methods and code optimization strategies will be implemented to minimize memory leaks and improve efficiency, particularly in high-concurrency environments.
5. **Distributed Systems and Networks**: To ensure *Draw It or Lose It* functions effectively across multiple platforms, a distributed system leveraging cloud services will be implemented. This system will allow multiple concurrent game instances to run smoothly. WebSockets will facilitate real-time communication between clients and the server, ensuring players receive immediate updates during gameplay. A fault-tolerant approach will be adopted, utilizing load balancers to distribute traffic evenly across servers, preventing downtime and ensuring a stable gaming experience even during peak usage.
6. **Security**: Security is a top priority for *Draw It or Lose It*. All communication between clients and servers will be protected using HTTPS to prevent interception attacks. Sensitive user data, including login credentials, will be encrypted using secure hashing algorithms. Additional access control mechanisms will also be enforced to ensure that only authorized users can access their respective sessions.