

Draw It of Lose It

# **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 4**

[**Domain Model 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/2023 | Joshua Perry | Initial version of the software design template. Included foundational sections such as Executive Summary, Design Constraints, Domain Model, and draft framework for Evaluation and Recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions (CTS) is tasked with the development of a web-based version of “Draw It or Lose It,” a game previously confined to Android applications. Our goal is to expand its accessibility across multiple platforms, retaining the game’s core features while adapting to a web environment. The game will involve multiple teams and players, with unique naming systems for teams and games to ensure easy identification and management. Our approach will focus on utilizing a client-server architecture, ensuring a scalable, maintainable, and user-friendly gaming experience. The solution will consider the need for a single instance of the game in memory, achieved through unique identifiers for game instances, teams, and players.

## Requirements

**Business Requirements:**

1. **User Engagement:** The application should be intuitive and engaging to maintain user interest.
2. **Scalability:** Capable of handling an increasing number of users and data without degradation in performance.
3. **Cost-Effectiveness:** Development and maintenance costs should be within the budgetary constraints set by the client.
4. **Market Compatibility:** The application should be compatible with current market trends and user preferences.

**Technical Requirements:**

1. **Platform Compatibility:** The application must be compatible across various platforms, including Windows, macOS, Linux, and mobile devices.
2. **Performance:** High responsiveness and low latency, especially during peak usage times.
3. **Security:** Robust security measures to protect user data and prevent unauthorized access.
4. **Data Management:** Efficient handling and storage of user data, ensuring data integrity and availability.

## [Design Constraints](#_2et92p0)

The primary design constraints include:

1. **Cross-Platform Compatibility:** The web-based game must function seamlessly across various platforms (Windows, macOS, Linux, and mobile devices)
2. **Concurrency Control:** Ensuring only one instance of the game exists in memory at any given time, requiring robust session and state management.
3. **Unique Identification:** **Implementing a system for unique game and team names, demanding a reliable database management system.**
4. **Performance:** Optimizing for real-time gameplay and rapid rendering of images, crucial for the game’s drawing and guessing elements.
5. **Scalability and Maintainability:** The architecture should support easy updates and scalability to accommodate an increasing number of users.

## [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 “Draw It or Lose It” includes the following classes:

1. **Game:** Represents an individual game session. Attributes include gameID, gameName, and a collection of Teams.
2. **Team:** Encapsulates details of a team, with attributes like teamID, teamName, and a collection of Players.
3. **Player:** Represents a player, with attributes such as playerID and playerName.
4. **Drawing:** Represents the drawings used as clues, with attributes for identification and linking to game rounds.

Relationships:

* A Game has multiple Teams.
* A Team consists of multiple Players.
* Drawings are associated with each Game, but not directly with Teams of Players.

Object-Oriented Principles:

* Encapsulation is used to protect the state of Game, Team, and Player objects.
* Inheritance could be utilized if a base class like ‘Entity’ is used for common attributes like IDs.
* Polymorphism might be applicable in future expansions, for example, in handling different types of game rounds or player roles.

**"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** | For hosting a web-based software application, Mac offers a stable and secure environment with excellent support for development tools. Its Unix-based system ensures reliability and compatibility with many server-side technologies. However, it can be more expensive due to hardware costs, and the user base for Mac servers is smaller compared to Linux and Windows. | Linux is widely favored for server-side applications due to its stability, security, and open-source nature, allowing for extensive customization. It supports a wide range of server technologies and is cost-effective. However, it requires more technical expertise to manage and configure compared to Windows. | Windows servers are known for their user-friendly interface and strong support for .NET technologies. They integrate well with other Microsoft products but can be costlier due to licensing fees. Windows servers may also be less flexible than Linux in terms of customization. | Mobile devices are generally not suitable for hosting web-based software applications due to their limited processing power, storage capacity, and network reliability. They are more suited as clients rather than servers in a distributed architecture. |
| **Client Side** | Developing for Mac clients requires consideration of the Apple ecosystem’s unique guidelines and standards. The cost of development can be high due to the need for specific hardware and software. Expertise in Swift or Objective-C is necessary, and the development process can be more streamlined but constrained with Apple’s ecosystem. | Linux, being open-source, offers a flexible development environment. It requires expertise in a variety of programming languages and environments. The cost can be lower, but the development time might increase due to the need to accommodate a wide range of distributions and configurations. | Windows development is supported by a wide range of tools and languages, with .NET being a prominent framework. The cost of development tools can vary, and there is a large talent pool available. The development time can be shorter due to the widespread familiarity and extensive documentation. | Developing for mobile devices involves considering different operating systems like iOS and Android. This required expertise in Jave/Kotlin for Android and Swift/Objective-C for iOS. The cost can be high due to the need for multiple development environments and testing on various devices. Time to market can be longer due to the complexity of mobile ecosystems. |
| **Development Tools** | Development on Mac primarily uses Xcode, with Swift and Objective-C as the main programming languages. Other tools include Visual Studio Code, Atom, and various web development environments. Mac is favored for iOS app development. | Linux supports a wide range of programming languages and IDEs, including Eclipse, PyCharm, and GCC for C/C++. It is particularly favored for open-source and server-side development. | Windows development is often centered around Visual Studio, supporting C#, .NET, and other Microsoft technologies. Other tools include Eclipse, IntelliJ IDEA, and SQL Server Management Studio. | For iOS, Xcode is the primary tool, whereas Android Studio is used for Android development. Other relevant tools include Unity for game development, Flutter, and React Native for cross-platform applications. |

## Recommendations

1. **Operating Platform**: Utilizing a combination of cloud computing services (e.g., AWS, Google Cloud, Azure) and web technologies (like HTML5, JavaScript) is ideal. This setup ensures broad compatibility across multiple devices and operating systems, including Windows, macOS, Linux, and mobile platforms.
2. **Operating Systems Architectures**:
   1. **Client-Server Architecture:** The client-side, built with HTML5 and JavaScript, offers a responsive and user-friendly interface. The server-side, powered by cloud services, manages game logic, data processing, and storage.
   2. **Cloud-Based Server Infrastructure:** Utilizes scalable, reliable cloud infrastructure to handle varying loads, ensuring smooth gameplay even during peak usage.
   3. **Cross-Platform Compatibility:** The web-based client ensures seamless operation across different operating systems, providing a consistent gaming experience.
3. **Storage Management**: A distributed database system like Amazon RDS or Azure SQL Database would be optimal. These systems provide scalability, high availability, and efficient data management, which are crucial for handling the game’s data requirements.
4. **Memory Management**:
   1. **Cloud Infrastructure Memory Management:** Cloud platforms typically employ advanced techniques like garbage collection, memory pooling, and dynamic allocation, offering efficient memory usage.
   2. **Client-Side Memory Management:** Browsers handle memory management on client devices, optimizing performance and resource utilization.
5. **Distributed Systems and Networks**:
   1. **Implementation:**
      1. **RESTful APIs and WebSocket:** These technologies will facilitate real-time communication between various platforms, ensuring smooth and synchronized gameplay.
      2. **Network Design Considerations:** Incorporating redundancy, load balancing, and failover strategies is crucial to maintain connectivity and handle potential outages.
6. **Security**:
   1. **Strategies:**
      1. **Secure Communication:** Implement HTTPS to secure data in transit.
      2. **Authentication:** Use OAuth or similar protocols for secure user authentication.
      3. **Data Encryption:** Encrypt sensitive data at rest in the database.
      4. **Compliance and Audits:** Regular security audits and adherence to standards like GDPR and CCPA are crucial for user data protection.
      5. **Cloud Security Features:** Leverage the security features offered by cloud providers to enhance overall security.