

Draw It or 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 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/10/2023 | Victor Udeh | First draft |

**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 aims to transform their 'Draw It or Lose It' Android game into a web-based application. Our goal is to develop an engaging, scalable, and efficient web-based version that expands the game's reach across multiple platforms while retaining its original charm and gameplay style. This document presents our approach to design and development, emphasizing the unique software requirements and constraints of this project.

## Requirements

Functional Requirements

1. **Multi-Platform Support**: The game must be accessible and fully functional across various platforms, including desktop and mobile devices.
2. **Unique Identification**: Each game session, team, and player must have unique identifiers to prevent confusion and duplication within the system.
3. **Team and Player Management**: The game must support the creation of multiple teams, with the ability to add multiple players to each team.
4. **Gameplay Mechanics**: Implement the core gameplay mechanics, including the rendering of images as clues, timing of rounds, and guess submission within the time limits.
5. **Leaderboard and Scoring**: The game should maintain a scoring system and display a leaderboard reflecting team performance.

Non-Functional Requirements

1. **Usability**: The game interface should be intuitive and user-friendly, ensuring ease of navigation and understanding for players of all ages.
2. **Scalability**: The application must be capable of handling a growing number of users and teams without degradation in performance.
3. **Performance**: Ensure minimal latency in gameplay, especially in rendering images and processing guesses.
4. **Security**: Implement standard security measures to protect user data and prevent unauthorized access.
5. **Network Efficiency**: Optimize for varying network speeds, ensuring the game remains playable even with fluctuating internet connectivity.

Design and Technical Requirements

1. **Cross-Browser Compatibility**: The web application should be compatible with major browsers like Chrome, Firefox, Safari, and Edge.
2. **Responsive Design**: The UI should be responsive and adapt to different screen sizes and resolutions.
3. **Technology Stack**: Specify the technologies to be used for frontend, backend, and database management.
4. **Data Storage and Retrieval**: Efficient handling of data storage, ensuring quick retrieval and update of game states, scores, and user information.
5. **API Integration**: If the game requires integration with external services (e.g., stock image libraries), the API integration should be seamless and secure.

## [Design Constraints](#_2et92p0)

The development of 'Draw It or Lose It' as a web application involves several constraints: Platform Compatibility (ensuring functionality across various devices), Unique Identification (for games and teams), Singleton Game Instance (maintaining a single game instance in memory), Scalability (accommodating a growing user base), and Network Dependence (relying on stable and fast internet connectivity). These constraints are pivotal in guiding our development strategy, focusing on cross-platform compatibility, efficient resource management, and robust network handling.

## [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 for 'Draw It or Lose It' demonstrates a well-structured game architecture. At its core is the **Entity** class, a base class for **Game**, **Team**, and **Player**. This hierarchy not only promotes code reuse and efficiency but also simplifies maintenance and scalability. The relationships between the classes are clearly defined, with **Game** aggregating **Teams**, and **Teams** consisting of **Players**. This reflects an organized, modular approach, essential for a dynamic, data-driven game environment.

The diagram's layout also reveals critical object-oriented principles, such as inheritance (shared attributes and behaviors in **Entity**), aggregation (composition of **Teams** in a **Game**), and encapsulation (each class managing its own data and behavior). These principles ensure that the software design is robust, flexible, and maintainable

**"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** | Mac servers, known for their stability and security, are well-suited for hosting web-based applications. They offer excellent support for various development languages and tools. However, they can be costlier than other platforms, and the availability of technical expertise might be more limited compared to Linux or Windows. | Linux is widely favored for server-side applications due to its robustness, flexibility, and open-source nature. It supports a wide range of programming languages and tools, making it a versatile choice. However, it may require more technical expertise to manage and configure compared to Windows servers. | Windows servers are known for their user-friendly interface and strong support from Microsoft, including regular updates and patches. They integrate well with other Windows-based systems and services. However, they can be more expensive in terms of licensing costs and may not offer the same level of flexibility as Linux. | Hosting a web-based application on mobile devices is not typical due to their limited processing power and storage compared to traditional servers. Mobile devices are more suited as clients in a web-based application architecture. |
| **Client Side** | Developing client-side software for Mac typically involves a focus on aesthetics and seamless integration with the macOS ecosystem. The development can be more expensive due to the need for specific tools and expertise, and the testing phase may require additional resources due to the variety of devices and OS versions. | Client-side development for Linux can be less costly due to the availability of open-source tools and resources. However, it requires expertise in Linux-specific technologies and consideration for the diverse range of Linux distributions. | Windows has a large user base, making client-side development on this platform a vital consideration. Development can be less costly due to the availability of various tools and resources. However, catering to all versions of Windows can add complexity. | Client-side development for mobile devices involves considering a wide range of device sizes, operating systems (iOS, Android), and capabilities. It requires specific expertise in mobile development and can be costly due to the need for extensive testing across different devices. |
| **Development Tools** | Development tools for Mac include Xcode for macOS-specific applications, along with popular IDEs like IntelliJ IDEA and Eclipse. Languages commonly used are Swift, Objective-C, and cross-platform languages like JavaScript and Python. | Linux supports a wide range of programming languages, including Python, Java, and C++. Popular IDEs include Eclipse, NetBeans, and Visual Studio Code. Linux’s open-source nature provides a broad range of tools and libraries for developers. | Windows supports various programming languages, with a focus on .NET languages like C# and VB.NET. Visual Studio is a prominent IDE for Windows development. Other languages like Java, Python, and JavaScript are also widely used. | Mobile development tools include Android Studio for Android apps, and Xcode for iOS apps. Common languages are Kotlin and Java for Android, and Swift and Objective-C for iOS. Cross-platform tools like React Native and Flutter are also popular. |

## 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**: A cross-platform web-based solution is recommended for expanding "Draw It or Lose It" to other computing environments. This approach allows the game to be accessible via web browsers on various devices, including desktops, laptops, tablets, and smartphones, regardless of the underlying operating system.
2. **Operating Systems Architectures**: The chosen architecture is based on web technologies like HTML5, CSS3, and JavaScript for the frontend, with a backend possibly using Node.js or a similar server-side technology. This architecture is independent of the client's operating system, providing flexibility and broad accessibility.
3. **Storage Management**: Cloud-based storage solutions, such as Amazon S3 or Google Cloud Storage, are recommended. These systems offer scalable, secure, and reliable storage, essential for a web-based application with potentially large numbers of users.
4. **Memory Management**: The recommended web-based platform primarily relies on the client device's browser for memory management. Modern browsers are efficient in managing memory for web applications, automatically handling tasks like garbage collection and memory allocation.
5. **Distributed Systems and Networks**: "Draw It or Lose It" can utilize a distributed architecture where the game's backend is hosted on cloud servers, ensuring scalability and reliability. Communication between different platforms is managed through web services and APIs, allowing seamless data exchange and gameplay across devices. Network connectivity and potential outages should be considered by implementing fallback mechanisms and offline functionalities.
6. **Security**: To protect user information, the application should implement HTTPS for secure communication, along with robust authentication and authorization mechanisms (like OAuth). Data encryption, both at rest and in transit, is crucial. Regular security audits and compliance with data protection regulations (like GDPR) will further enhance user data security.