

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.3

## 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 | 07 /23/23 | Simon Byelkin | First version of the document. |
| 1.1 | 08/06/23 | Simon Byelkin | Revision of the document, confirmed that all sections are completed. |
| 1.2 | 08/20/23 | Simon Byelkin | Revision of the document, confirmed that the section recommendations is accurately filled out. |
| 1.3 | 08/20/23 | Simon Byelkin | Final revision. All sections filled out correctly. |

**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 task at hand is to transform "Draw It or Lose It," an Android-only game, into a web-based application for broader accessibility. The solution involves developing a web-based version of the game, ensuring it is compatible across multiple platforms, including desktops, laptops, and mobile devices.

Key features of the original game will be retained, such as support for multiple teams and players, and unique identifiers for each game, team, and player. Software design patterns like Singleton and Iterator will be used to manage game instances.

The client should be prepared for iterative testing and feedback during development, and subsequent hardware requirements determination based on the software decisions. The goal is to increase user engagement and revenue by making the game accessible to a wider audience.

## Requirements

Business Requirements:

1. Broaden Accessibility: The game, currently only available as an Android app, should be made accessible across multiple platforms, including desktops, laptops, and various mobile devices.
2. Maintain Game Structure: The core structure of the game, where teams compete to guess what is being drawn, should be retained in the web-based version.
3. Unique Identifiers: Each game, team, and player should have a unique identifier to ensure smooth operation and user experience.

Technical Requirements:

1. Cross-Platform Compatibility: The game should be developed as a web-based application, ensuring compatibility across various devices and operating systems.
2. Singleton Design Pattern: Only one instance of the game should exist in memory at any given time. This can be achieved using the Singleton design pattern.
3. Iterator Design Pattern: The game and team names should be unique. The Iterator design pattern can be used to check whether a name is already in use when a user is choosing a team name.
4. Base Entity Class: A base class called Entity should be created to hold common attributes and behaviors. Other classes like Game, Player, and Team should inherit from this Entity class.

## [Design Constraints](#_2et92p0)

1. Cross-Platform Compatibility: The game must be designed to be compatible across various platforms, including desktops, laptops, and mobile devices. This requires the use of a responsive design that adapts to different screen sizes and operating systems. The constraint here is ensuring a consistent user experience across all platforms.
2. Single Instance in Memory: The game must be designed to have only one instance in memory at any given time. This constraint, managed using the Singleton design pattern, ensures system resources are used efficiently.
3. Unique Identifiers: Each game, team, and player must have a unique identifier. This constraint, managed using the Iterator design pattern, ensures smooth operation and user experience but requires a robust system for generating and tracking unique identifiers.
4. Web-Based Distributed Environment: The game must be developed for a web-based distributed environment. This constraint requires careful consideration of network latency, server capacity, and data synchronization to ensure a smooth and responsive user experience.
5. Inheritance from Entity Class: The Game, Player, and Team classes must inherit from a base Entity class. This constraint simplifies the code structure and promotes code reuse but requires careful design to ensure correct and efficient inheritance.

## [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 Entity class is the base class for Game, Team, and Player classes. This demonstrates the object-oriented principle of inheritance, where child classes inherit properties and methods from a parent class. This allows for code reusability and efficiency, as common attributes and behaviors are defined only once in the Entity class.

The GameService class uses the Singleton design pattern, ensuring that only one instance of GameService exists at any given time. This is a crucial requirement for the game application. The Singleton pattern is an object-oriented design pattern that restricts the instantiation of a class to a single instance.   
GameService class is associated with the Game class, the Game class is associated with the Team class, and the Team class is associated with the Player class. This shows the principle of association, where classes are linked to each other in a hierarchy. In this case, a game service can have multiple games, a game can have multiple teams, and a team can have multiple players.

The ProgramDriver class uses the SingletonTester class, demonstrating the principle of dependency. This means that the ProgramDriver class depends on the SingletonTester class for some functionality.

**"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 are known for their stability and seamless integration with other Apple devices, but they can be expensive and may not be as flexible as other options. | Linux servers are highly customizable, stable, and efficient. They are widely used in web hosting due to their open-source nature and cost-effectiveness. However, they require more technical expertise to set up and manage. | Windows servers are user-friendly and integrate well with other Microsoft products. They support .NET framework, which can be crucial for certain applications. However, they can be more expensive than Linux servers. | Mobile devices are typically not used for hosting web-based software applications due to their limited resources and potential security issues. |
| **Client Side** | Developing for Mac clients requires consideration of the unique macOS interface and user expectations. It may require more time and expertise to ensure compatibility with the latest macOS versions. Also, Apple Store is very demanding, if you want to add your application there. | Linux clients are diverse, and supporting multiple Linux distributions can increase development time and cost. Open-source tools can help reduce costs. However not a lot of end users use Linux, it is mostly used by advanced users. | Windows clients make up a large market share, so supporting them is usually essential. Development might be more straightforward due to the wide range of tools and resources available. | Mobile devices require a responsive design to ensure the application looks good on various screen sizes. Development for mobile devices can be complex due to the need to support multiple operating systems (iOS, Android). IOS may be easier to develop a more responsive application but Android has more users. |
| **Development Tools** | For Mac, you might use Xcode as an IDE, with languages like Swift or Objective-C for native applications. Or Mac OS is a versatile developer OS offering all the same capabilities Linux does. VSCode being the best code editor with its wide support for extensions. | For Linux, tools like VSCode, Eclipse, NetBeans, or IntelliJ IDEA, Vim can be used, with languages like Python, Java, or C++. For web applications, JavaScript(TypeScript), HTML, and CSS are commonly used. | For Windows, Visual Studio is a popular IDE, with languages like C#, VB.NET, or C++ for native applications. For web development Windows also supports same tools Linux and Mac does, and also supports VSCode. Also Windows has WSL and can run Linux binaries directly. So it allows to use all the same tools Linux has, with great integration with VSCode(CLI tool `code `. | For mobile devices, you might use Swift or Objective-C for iOS (with Xcode as an IDE), Java or Kotlin for Android (with Android Studio as an IDE), or JavaScript, HTML, and CSS for web applications. Cross-platform tools like React Native or Flutter can also be used. |

## 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**: I recommend using a cross-platform web-based approach for expanding "Draw It or Lose It" to other computing environments. This approach allows the game to be accessed from any device with a web browser, including Windows, Mac, Linux, and mobile devices.
2. **Operating Systems Architectures**: The web-based approach is platform-independent and relies on web technologies like HTML, CSS, and JavaScript. The server-side can be built using Node.js, a JavaScript runtime built on Chrome's V8 JavaScript engine, which is efficient and scalable. A web framework like NextJS 13 combined with other JS libraries like ThreeJS can be used across the stack. NextJS allows to write API endpoints in TypeScript and frontend using TSX format being an advanced version of React. ThreeJS would handle 3d animations for the game.
3. **Storage Management**: For storage management, a cloud-based solution like Amazon Web Services (AWS) or Google Cloud Platform (GCP) can be used. These platforms offer scalable and secure data storage options, including relational databases for structured data and NoSQL databases for unstructured data.
4. **Memory Management**: In a web-based platform, memory management is handled by the JavaScript engine in the user's browser on the client side, and by Node.js on the server side. Both automatically allocate and deallocate memory as needed, reducing the risk of memory leaks.
5. **Distributed Systems and Networks**: The game can be designed as a distributed system where the client-side code runs on the user's device and communicates with the server-side code over the internet. This can be accomplished using HTTP/HTTPS protocols. Dependencies between components can be managed using APIs and microservices architecture. However, network connectivity and outages must be considered, and appropriate error handling and retry logic should be implemented.
6. **Security**: User information can be protected using encryption during transmission (HTTPS) and at rest (database encryption). User authentication can be implemented using secure techniques like OAuth or JWT. The recommended platforms, AWS and GCP, offer built-in security features to protect data and manage user access. Regular security audits and updates should be part of the development and maintenance process to ensure ongoing security.