

# 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 4**](#_Toc115077324)

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

[**Recommendations 6**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/13/2024 | Raul Rodriguez | Changes were made to the cover page, the document revision history, the executive summary, design constraints, system architecture view, domain model and recommendation |
| 1.1 | 11/28/2024 | Raul Rodriguez | Updated system architecture view and recommendations. |
| 1.2 | 12/9/2024 | Raul Rodriguez | Updated the Recommendations section to align with rubric requirements. |

**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 wants to expand its game, *Draw It or Lose It*, from an Android app to a web-based game that works on many devices. This design plan explains how the game will meet important needs, like allowing multiple teams and players, making sure names and IDs are unique, and having only one active game at a time. The design uses simple programming ideas, like reusing code and organizing it well, to make the game easy to update and work smoothly. This plan will help The Gaming Room create a safe, reliable, and user-friendly game that works on different platforms.

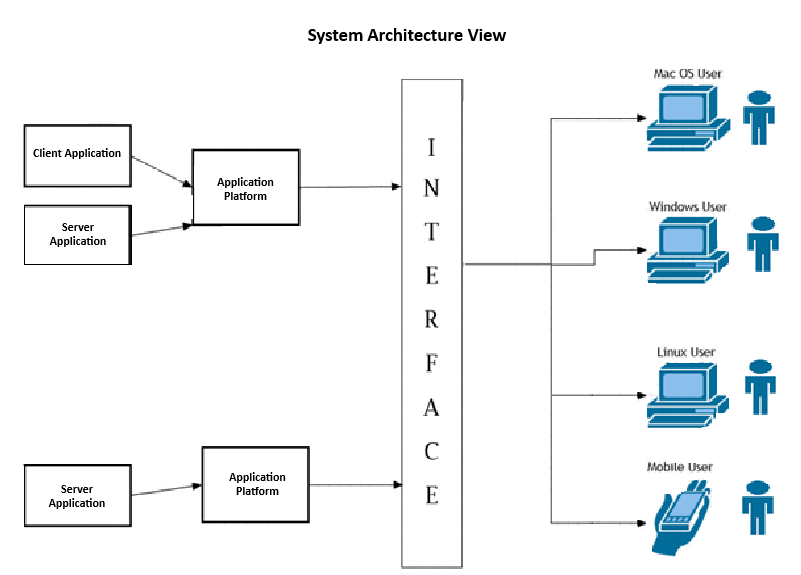
## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* **Web-Based Platform**: The game must work on a web-based platform, ensuring compatibility across devices and browsers while addressing network and performance challenges.
* **Unique Identifiers**: Each game, team, and player must have unique IDs and names to prevent duplication and conflicts.
* **Single Active Game**: Only one game instance can run at a time to ensure proper resource management and prevent errors.

## [System Architecture View](#_ilbxbyevv6b6)



## [Domain Model](#_8h2ehzxfam4o)

The UML diagram shows how the Draw It or Lose It game is organized, with different classes and how they work together. At the center is the Entity class, which is the base for all other classes like Game, Team, and Player. The Entity class has shared features like id and name to make sure each game, team, or player has a unique identifier and name. This setup helps reuse code and keep everything consistent.

The Game class comes from the Entity class and manages multiple Team objects. Similarly, the Team class also comes from Entity and keeps track of many Player objects. This structure makes it easy to handle many games, teams, and players while keeping everything organized.

The GameService class is important because it manages all the games and makes sure only one game is running at a time using the Singleton Design Pattern. This helps prevent errors and ensures everything runs smoothly. The GameService also has methods to add games, teams, and players, making it the main manager for the system. It uses encapsulation to protect important data, so only certain methods can access or change it.

The ProgramDriver class is where the program starts, creating and using the GameService instance. It works with the SingletonTester class to confirm that the Singleton pattern is working. This setup makes sure responsibilities are split clearly: the ProgramDriver starts the program, and the GameService handles the system’s logic. The GameService also hides complex details about managing games, teams, and players, making it simple for other parts of the program to use. Together, these features make the system organized, easy to use, and scalable.

**"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)

I revised the recommendations and added a system architecture view to provide a comprehensive plan for the game’s functionality and scalability. The client-side interface will benefit from using adaptable technologies such as JavaScript, HTML, and CSS, while the server side can utilize reliable frameworks like Node.js or Django to manage backend operations. To address hosting needs, cloud solutions like AWS or Google Cloud emerged as strong contenders due to their ability to scale resources dynamically and ensure consistent performance. I also considered the use of peer-to-peer (P2P) networking as an alternative, which is popular in games like Roblox and Minecraft. However, due to the heightened risk of security vulnerabilities and the complexity of maintaining such a network for large-scale use, P2P may not be the most suitable option for this project. Overall, these recommendations focus on using robust, scalable, and secure solutions that align with the project’s goals while optimizing both cost and efficiency.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macs are secure and reliable but more expensive to use for hosting. They require extra setup compared to other options. | Linux is cheap, secure, and widely used for hosting but needs technical skills to manage. | Windows servers are easy to use and work well with Microsoft tools but can be costly and less flexible. | Mobile devices can’t host web apps but connect to servers like Linux, Windows, or Mac. |
| **Client Side** | Developing for Mac requires tools like Xcode and Swift, which may increase time and costs if the team isn’t familiar. | Linux is cost-effective but needs developers with Linux experience, which could take extra time. | Windows is developer-friendly, with tools like Visual Studio, making it faster and more affordable to support. | Developing for mobile needs tools like Flutter or React Native to work on both Android and iOS, but it can take more time and cost more. |
| **Development Tools** | |  | | --- | |  |  |  | | --- | | Mac development uses tools like Xcode and IntelliJ, but these require macOS, limiting flexibility. | | Linux supports tools like Eclipse and VS Code, which are free but may need more setup. | Windows tools like Visual Studio and Eclipse are widely used and simple to set up. | Mobile development requires tools like Android Studio or Xcode, or cross-platform tools like Flutter for faster development. |

## 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 web-based platform is the best choice for expanding Draw It or Lose It. This will let players access the game through web browsers on desktops, tablets, and smartphones. It also makes it easier for players to join without downloading or installing anything. A web-based approach supports compatibility across multiple devices, so the game can reach more users.
2. **Operating Systems Architectures**: The recommended setup is a client-server architecture. The client side will use technologies like HTML, CSS, and JavaScript to create a smooth and responsive design for players. The server side will handle important tasks such as saving player data, running the game logic, and managing all game processes. This setup ensures the game runs efficiently and keeps everything organized.
3. **Storage Management**: A mix of a relational database and cloud storage is ideal for managing game data. The database will store essential details like player accounts, scores, and game stats, while cloud storage will handle larger files, such as images and game assets. This ensures the system is scalable, secure, and accessible across devices.
4. **Memory Management**: The platform will use memory tools built into modern web browsers. These tools will automatically allocate and free up memory to keep the game running smoothly. This reduces the chance of issues like lag or memory leaks, ensuring a fast and reliable gaming experience.
5. **Distributed Systems and Networks**: To allow players to connect and play from anywhere, the game will use a central server hosted in the cloud. This setup ensures real-time updates like player actions and scores are handled efficiently. It also reduces the impact of slow connections or minor outages, so players have better and consistent experience.

**Security**: Security is critical for protecting user data. All data sent between devices will be encrypted, and user authentication (such as secure login systems) will ensure accounts are safe. These features will prevent unauthorized access and protect sensitive information, making the platform secure and trustworthy.