

<Name-of-Software-Application>

# **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 |
| --- | --- | --- | --- |
| 2.0 | <11/15/24> | <Ben Conaway> | <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](#_sbfa50wo7nsh)

This document explains the plan for creating a web-based game called "Draw It or Lose It" for our client, The Gaming Room. The game is based on the classic TV show \*Win, Lose, or Draw\*, where teams try to guess what is being drawn. The game will use images from a library of stock drawings as clues and let multiple teams play in four rounds. By making the game available on the web, it will be accessible on different devices and offer a better overall experience. The design takes the client’s needs into account and aims to create a fun and easy-to-use game.

## Requirements

The game must be accessible through a web platform that works on different devices and operating systems. Each game should allow one or more teams to play, and each team can have multiple players. Team and game names must be unique to avoid conflicts, and users should be able to check if a name is available when creating a team.

Only one instance of the game should be running at any time. This can be done by using unique IDs for each game, team, and player. Each round should have a set time limit, like one minute, with the drawing gradually revealed until it’s fully shown at the 30-second mark. If a team doesn’t guess the drawing in time, the other teams can each make one guess within a 15-second limit to try to solve it.

## [Design Constraints](#_2et92p0)

**Web-Based Platform:** The game must be designed for the web, which comes with challenges like network connections, security, and making sure it works on different web browsers and devices.

**Unique Names:** The system should make sure that game, team, and player names are unique to avoid any confusion and to make it easier for users to create and join games.

**One Game Instance:** The design must ensure that only one instance of the game is running at a time to make sure it works correctly.

## [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 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.**

The diagram above shows the different parts of the "Draw It or Lose It" game application. The Entity class is the main class that other classes are built from. It includes basic features like an 'id' and a 'name' that are shared by all the other classes. This means that every object in the game, whether it's a game, team, or player, will have an ID and a name.

The Game, Team, and Player classes are all based on the Entity class. These classes represent the key parts of the game: a Game has multiple Teams, and each Team has multiple Players.

The GameService class is in charge of managing the games. It has a special connection to the Game class, meaning it helps keep track of all the games happening. Similarly, the Game class manages the Teams, and the Team class manages the Players.

The ProgramDriver class is where the game starts. It holds the main function and serves as the entry point for the application. Inside ProgramDriver, the GameService class is set up to be a singleton, which means only one instance of it can exist throughout the whole application. ProgramDriver is responsible for adding games, teams, and players by using GameService. It also depends on the SingletonTester class, as shown by the <<uses>> arrow.

The diagram shows important concepts of object-oriented programming, such as inheritance, encapsulation, and abstraction. For example:

**Inheritance:** The Entity class is the parent of the Game, Team, and Player classes, meaning these classes can share basic features from Entity, like 'id' and 'name', which helps reduce repeating code.

**Encapsulation:** The GameService class hides details like its constructor and the list of games, only letting users interact with the game through specific methods. This keeps the data safe and prevents mistakes.

**Abstraction:** The classes focus on what the game, teams, and players do, rather than showing all the complicated details of how they work behind the scenes. This makes the game easier to use and understand.

## [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** | Unix-based systems provide a stable and secure foundation for hosting web applications. They have a strong ecosystem and are good for developers. However, they tend to have higher hardware costs and are less scalable than Linux and Windows. | An open-source operating system that lets you customize and adapt it, with many software and tools available. It's very scalable, stable, and secure. However, it has some issues with its graphical interface and may not work with all hardware. | It supports a wide variety of software and has a strong community of developers. It works with a broad range of hardware and offers solid support and documentation. However, it is more likely to have security vulnerabilities. | Portable with touchscreen and gesture controls, but limited by small screen sizes. Hardware capabilities can vary. |
| **Client Side** | The simple and easy-to-use interface makes it easier to learn. However, creating and maintaining multiple versions of the client can raise development costs, extend timelines, and require a range of specialized skills. | It's free to use and share, but you should keep in mind other costs like hardware and tools. It can be harder to learn, and different skills are needed for different clients. | Licensing fees might be more expensive than those of open-source options. | Key factors to consider are responsive design and connectivity restrictions. Native features like the camera, GPS, and push notifications are also important. |
| **Development Tools** | Node.js and JavaScript are widely used, along with integrated development environments (IDEs) like VSCode and XCode. | The extensive ecosystem features IDEs like VSCode, Atom, and Sublime Text. It also offers a powerful command-line interface and package management tools such as apt and yum. | C# and the .NET framework are commonly used to develop Windows-based web applications. Popular IDEs for this include Visual Studio and JetBrains. | In addition to Java and JavaScript, Kotlin, Swift, and Objective-C are also used. Development tools include Android Studio, XCode, and device emulators or simulators. |

## 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**: To help the Gaming Room expand \*Draw It or Lose It\* to different computing environments, it’s recommended to switch to a web-based platform. This would make the game accessible on any device with a web browser, including desktops, laptops, tablets, and smartphones. Using web technologies would allow the game to reach a larger audience and ensure a consistent experience across various platforms.
2. **Operating Systems Architectures**: For the selected web-based platform, the system architecture will focus on client-server and web technologies. On the client side, HTML, CSS, and JavaScript will be used to display the game interface and manage user interactions. On the server side, a multi-layered architecture will be implemented, including separate layers for presentation, application logic, and data management.
3. **Storage Management**: An effective storage management solution would combine a relational database management system (RDBMS) with cloud storage services. The RDBMS would manage structured data like game progress, user profiles, and game statistics, while cloud storage would be used to store media assets, such as stock images, offering scalability and easy access across different platforms.
4. **Memory Management**: The suggested web-based platform takes advantage of the automatic memory management features offered by modern web browsers. These browsers use a garbage collection system to manage memory allocation and deallocation, so developers don't need to handle it manually. This improves memory efficiency and helps prevent memory leaks.
5. **Distributed Systems and Networks**: To allow communication across different platforms, the game can adopt a distributed software architecture and use network connectivity. This can be done by setting up a central server or cloud-based infrastructure that acts as a hub for communication between game clients. The server would manage game synchronization, provide real-time updates, and handle message exchange between players on different devices. The system should also be designed to handle network issues like occasional outages or slow connections, with built-in error handling and synchronization features.
6. **Security**: To ensure the security of user data across different platforms, a range of security measures can be put in place. The chosen web-based platform should support secure communication protocols. User authentication methods, like usernames and passwords, can be used to control access to game features and profiles. Furthermore, encryption techniques can be applied to protect sensitive user information, both when stored in databases and during transmission over the network.