

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 0.1 | 01/27/2024 | Stephen Chryn | Initial Revision |
| 0.2 | 02/11/2024 | Stephen Chryn | Expanded on Recommendations |
| 0.3 | 02/23/2024 | Stephen Chryn | Fixed formatting |
| 0.4 | 02/23/2024 | Stephen Chryn | Elaborated on Design Constraints |
| 1.0 | 02/24/2024 | Stephen Chryn | Finalized 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](#_heading=h.35nkun2)

"Draw It or Lose It" is an interactive web-based game application inspired by the 1980s game show "Win, Lose or Draw." The game involves teams competing to guess phrases or titles based on a sequence of stock drawings. This document outlines a software design approach to develop this application, focusing on the requirements specified by the client. The application will be developed using Java, a versatile programming language ideal for creating scalable and maintainable web applications.

## Requirements

1. Game Server: A central server to manage game sessions, team interactions, and drawing sequences.
2. Client Interface: A user interface, whether web-based or which can speak to the main server, for players to interact with the game.
3. Database Management: A system to handle unique names for games, teams, and players, as well as storing game states. While this can be done within the runtime memory, it is recommended to save some session data to a central database in order to keep track of player, game and instance data.

## [Design Constraints](#_heading=h.1ksv4uv)

Developing "Draw It or Lose It" as a web-based application in a distributed environment imposes several design constraints that significantly impact the application development process.

Real-time interaction is a critical aspect of the game. Network latency can affect the synchronicity of the drawing reveals and player responses. Optimizing data transmission and implementing efficient real-time communication protocols are essential to mitigate latency issues. We should utilize lightweight data formats, like JSON, compress data where possible and implement a client-side experience to handle minor network delays gracefully. This can be done by serving all necessary information for the client upfront, with the client only needing to contact the server when requesting cross-client information as well as non-session based information.

Players should be able to access the game from various devices and browsers, which requires the application to be responsive and compatible across different platforms. If the client interactions will be done through a browser then we need to check for cross compatibility between the browsers. The “most used” browsers are Chromium based but Safari's use of WebKit and Firefox’s use of Gecko means that there might need to be some changes we’ll have to make when dealing with the different browser engines. If we create a client interface that is based on Java that can still interact with the server there will be less requirements when it comes to cross platform, due to Java's "write once, run anywhere" approach using their Java Virtual Machine.

## [System Architecture View](#_heading=h.44sinio)

The system architecture for a game like "Draw It or Lose It" must be meticulously planned to ensure a seamless and engaging experience for the players, whether it is browser-based or Java-based on the client side.

Browser Based Architecture:

1. Front-End requirements:

A responsive and intuitive web interface using HTML, CSS, and JavaScript frameworks like React or Vue or libraries such as HTMX. JavaScript or TypeScript for handling user interactions, game timers, and dynamic rendering of drawings. WebSockets or similar technologies for real-time updates of game state and player interactions. As stated in the previous section, cross-browser and cross-platform compatibility to ensure a consistent experience on different devices and browsers.

1. Back-End requirements:  
   Along with the game driver there will need to be a backend of the browser server to serve as a bridge between the client's browser and the game server. This can be done using Node.js, Java Spring Boot, or any others. This will also be what’s used to feed the frontend to the client. This will also utilize a database system, like PostgreSQL, MongoDB, for storing game data, user profiles, and unique names.

## [Domain Model](#_heading=h.2jxsxqh)

ProgramDriver:

* This is the main class with a main() method that runs the application. It uses the SingletonTester class.

SingletonTester:

* This class contains a method testSingleton() which tests the singleton pattern implementation of GameService.

GameService:

* It is a singleton, as indicated by the presence of a getInstance() method and a private constructor.
* It has a one-to-many relationship (1..\*) with the Game class, meaning a single GameService can manage multiple games.
* It holds three private fields to keep track of the next IDs to be used for games, players, and teams.
* It also maintains a list of games.

Game:

* Inherits from Entity, meaning it has an ID and a name.
* It has a one-to-many relationship with Team, as a game can consist of multiple teams.
* It contains methods to add teams and a toString() method to return a string representation.

Team:

* Inherits from Entity as well.
* It has a one-to-many relationship with Player, indicating a team consists of multiple players.
* It includes methods to add players and a toString() method.

Player:

* Also inherits from Entity.
* Has an ID and a name, and a toString() method.

Entity:

* It is a base class with common attributes id and name, which are inherited by Game, Team, and Player.
* It provides common methods like getId(), getName(), and toString() that are available to all subclasses.

**"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](#_heading=h.z337ya)

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** | Known for their performance and strong security features, Mac is good for hosting web-based applications, though they can be cost-prohibitive and offer limited compatibility with some software stacks compared to more widely-used server environments like Linux or Windows. | Renowned for its stability, flexibility, and open-source nature, Linux excels in hosting web-based applications due to its wide compatibility and cost-effectiveness, but can have a steeper learning curve and require more hands-on management compared to other operating systems. | With its user-friendly interface and extensive support for various software, Windows is advantageous for hosting web-based applications that require specific Windows-based technologies, but it may lag behind in terms of cost, security, and resource efficiency compared to Linux-based servers. | Running a web-based server on a mobile device, while not sustainable, is not impossible. Due to their less powerful hardware, constrained resources, and potential network limitations compared to traditional server environments, mobile devices are generally limited in hosting capabilities for web-based applications. |
| **Client Side** | Supporting multiple types of clients on Mac requires consideration of higher development costs, potentially longer development time due to the need for specialized expertise, and adherence to Apple's specific software standards and ecosystem. | Supporting multiple client types on Linux necessitates a focus on cost-efficiency, leveraging open-source tools and community support, while also accounting for possibly extended development time and the need for specialized expertise in Linux environments and diverse software stacks. | Developing for multiple client types on Windows often involves Microsoft integration, utilizing widely available development tools and resources for potentially faster development, and requiring expertise in Windows-specific technologies and compatibility considerations. | Supporting various clients on mobile devices demands a focus on responsive design, potentially lower profit due to App Store taking cuts and time due to the need for expertise in diverse mobile platforms and testing on multiple devices, along with adhering to different app store guidelines and requirements. |
| **Development Tools** | For deployment on Mac, relevant programming languages include Swift and Objective-C, with xcode being Apple's native IDE, alongside other utilities such as Homebrew and Terminal for advanced tasks. | For developing software for deployment on Linux, popular programming languages include Python, Java, and C++, with tools like Eclipse, Visual Studio Code, on an Interface and Nano, Neovim, and GCC in a terminal. Alongside a variety of terminal-based utilities and package managers like APT, Pacman, DNF, and YUM. | For software development targeting Windows deployment, commonly used programming languages include C++, Python, C#, .NET, and VB, and many other coding languages developed by Microsoft. With tools such as Visual Studio, VSCode, and PowerShell, and utilities like Windows SDK. | Mobile device software development typically involves languages like Swift and Objective-C for iOS, Java for Android, using IDEs such as xcode for iOS and Android Studio for Android, along with tools like Unity for cross-platform 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**: Linux is an appropriate operating platform for The Gaming Room to expand "Draw It or Lose It" across various computing environments and server-based solutions. Linux being a free, open source, solution means that updating hardware and cloning systems will come with no strings attached, while expanding with Windows can lead to lengthy calls with Microsoft IT that tend to result in buying new keys for upwards of $200. Linux is also a better choice over MacOS as MacOS doesn’t really offer a Server solution. Which means in every server station The Gaming Room will need to be run on a normal Desktop Environment.
2. **Operating Systems Architectures**: Linux operates on an open-source, x86\_64 architecture that is highly customizable and supports a wide range of hardware, making it ideal for diverse computing environments from personal computers to servers. While both of the competitors thrive with x86\_64 architecture, Linux uses the architecture more efficiently, while being able to customize every aspect of use.
3. **Storage Management**: For the recommended Linux operating platform, a scalable storage management system like MySQL or PostgreSQL would be perfect, offering reliability, high performance, and compatibility with various data-driven applications. While MySQL, developed by Oracle, may have more documentation to it, PostgreSQL being an open source project might have more community development and stability. Personally, I’ve found that PostgreSQL is easier to use as if you ever need to work with 3rd software. If a bit of software is specifically designed to only work with 1 database, that database tends to be PostgreSQL, while if a bit of software is designed to work with MySQL they usually give it cross compatibility with PostgreSQL.
4. **Memory Management**: Linux efficiently uses advanced memory management techniques such as virtual memory, demand paging, and swap space, which would optimize the performance and scalability of the "Draw It or Lose It" software by effectively managing and allocating system resources.
5. **Distributed Systems and Networks**: To enable "Draw It or Lose It" to communicate across various platforms in a distributed software environment, a combination of well-designed APIs, networking protocols, and strategic handling of dependencies and potential network issues is essential.
6. **Security**: To protect user information in "Draw It or Lose It" across various platforms, especially within a Linux-based environment, several security measures should be implemented. Utilize TLS/SSL encryption for all data transmitted over the network to protect data in transit from eavesdropping or interception. Additionally, sensitive data stored in databases should be encrypted at rest. Implement authentication mechanisms, such as OAuth2 (Using Google, Microsoft, or any other OAuth provider) or JSON Web Tokens, to securely manage user identities and sessions. Implement fine-grained authorization controls to ensure users can only access the data and features they are permitted to.