

<Draw It or Lose It>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc182667557)

[Table of Contents 2](#_Toc182667558)

[Document Revision History 2](#_Toc182667559)

[Executive Summary 3](#_Toc182667560)

[Design Constraints 3](#_Toc182667561)

[System Architecture View 4](#_Toc182667562)

[Domain Model 4](#_Toc182667563)

[Evaluation 5](#_Toc182667564)

[Recommendations 7](#_Toc182667565)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/16/24 | Christopher Phillips |  |

**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)

**Executive Summary**

The Gaming Room, a client of Creative Technology Solutions (CTS), seeks to develop a web-based version of their popular Android-only game, *Draw It or Lose It*. The goal is to expand the game’s availability across multiple platforms while maintaining functionality and enhancing user experience.

To address the client’s needs, CTS has been hired to design and implement the development environment and create an application capable of serving multiple clients and platforms. The game will feature an extensive library of stock drawings, displayed as clues, with images rendering at a steady rate to ensure smooth gameplay.

The application will incorporate the following requirements:

* A game will involve one or more teams, each consisting of multiple players.
* Game and team names must be unique, allowing users to verify name availability when choosing a team name.
* Only one instance of the game can exist in memory at any time, achieved by leveraging unique identifiers for games, teams, and players.

Object-oriented programming principles and design patterns will be used to develop a flexible, scalable, and secure application. By utilizing the singleton pattern, the solution will prevent duplicate instances of the game service. The iterator pattern will ensure unique naming conventions across games and teams.

This approach simplifies development, facilitates easier maintenance, and establishes a robust framework for future updates. CTS is confident this solution will meet the client’s technical requirements and enable *Draw It or Lose It* to launch successfully across multiple platforms.

## [Design Constraints](#_2et92p0)

Developing *Draw It or Lose It* as a web-based application introduces several design constraints that need to be considered for successful implementation:

* **Performance and Scalability**  
  The application must efficiently manage multiple concurrent users and teams while maintaining performance. Additionally, the game will require more storage to accommodate the extensive library of stock images and sufficient memory to render these images at the required speed, ensuring performance meets or exceeds that of the current mobile application.
* **Security Considerations**  
  The security structure of the application will differ significantly from the mobile version. While the mobile app relies on the device's inherent security features, the web-based version must implement more robust security measures. This includes advanced authentication methods to verify users and protect user information within a client-server environment.
* **Unique Naming Requirements**  
  Game and team names must be unique, requiring mechanisms for users to check name availability when creating new teams.
* **Single Instance Constraint**  
  Only one active instance of the game may exist in memory at a given time. This will be achieved using unique identifiers for games, teams, and players.
* **Rendering and Resource Management**  
  The game must efficiently select and display images from a large library of stock drawings, ensuring smooth rendering and gameplay experiences.

## [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 class diagram represents the structure and relationships between the entities in the Draw It or Lose It application. The diagram adheres to core object-oriented programming (OOP) principles, ensuring modularity, reusability, and scalability

**Classes and Relationships**

* The Entity class serves as the base class for Game, Team, and Player, encapsulating shared attributes (id and name) and behaviors. This approach promotes code reusability and reduces redundancy.
* The GameService class functions as the primary manager for the game’s lifecycle and uses the Singleton pattern to ensure only one instance exists in memory. It contains a list of active games and provides methods to add, retrieve, and count games.
* The Game class represents an individual game instance, holding a list of Team objects. Each game can add teams and ensure their uniqueness.
* The Team class represents a group of players within a game and manages a list of Player objects.
* The Player class encapsulates the individual player data and inherits common properties from Entity.

**Object-Oriented Programming Principles**

* Encapsulation: Each class protects its data by using private attributes and provides controlled access through public methods.
* Inheritance: The Entity base class ensures all shared attributes (id and name) are inherited by Game, Team, and Player, streamlining the design.
* Polymorphism: The base Entity class can be extended or overridden in the future to support additional functionality for specific entities.
* Abstraction: By abstracting common behaviors in the Entity class, the design reduces complexity and enhances maintainability.

**Efficiency in Meeting Software Requirements**

The application requirements specify:

1. A game will involve one or more teams.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique, allowing users to check if a name is in use when choosing a team name.
4. Only one game instance can exist in memory at any given time.

The design effectively ties the domain model to these requirements:

* The Entity object serves as a base for other classes to inherit, ensuring each class has a unique identifier and name. This implementation satisfies requirement 3 by enabling uniqueness for game and team names.
* The Iterator pattern is applied to handle team and player management, fulfilling requirements 1 and 2 by ensuring seamless interaction with teams and their associated players.
* The Singleton pattern is utilized in the *GameService* class to enforce the single-instance constraint, addressing requirement 4 by ensuring that only one instance of the game exists in memory at any time.

These design choices ensure efficient and scalable handling of software requirements while maintaining data integrity and aligning closely with the client’s needs.

**"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** | macOS provides robust stability, security, and a user-friendly interface for development and hosting. It supports UNIX-based environments but requires licensing. While virtualization is supported, macOS hardware is expensive and limits accessibility for smaller organizations. | Linux is open-source and highly flexible, making it ideal for server environments. It offers exceptional virtualization support (e.g., Docker, Kubernetes) and strong security through customizable user access controls. Requires expertise to configure effectively but is cost-efficient for development. | Windows is widely used for commercial servers and offers excellent GUI-based tools. However, it is more vulnerable to malware compared to UNIX-based platforms. Virtualization options like Hyper-V make it versatile for development and testing, but licensing and costs are higher. | |  | | --- | | Mobile devices are not suitable for server hosting or development environments. Instead, they are consumer-focused, with server-side interactions relying on UNIX-based (Linux or macOS) or Windows server platforms. |  |  | | --- | |  | |
| **Client Side** | macOS supports Safari and Chrome browsers, offering smooth performance for client-side use. Safari is optimized for macOS, but Chrome has broader cross-platform compatibility. Despite its stability, macOS has a smaller user base compared to Windows. | |  | | --- | | Linux offers flexibility for client-side use, supporting browsers like Firefox and Chrome. While Firefox is a popular choice for Linux, the overall Linux desktop user base is smaller. Cross-platform compatibility with modern web standards makes it a strong option for niche development. |  |  | | --- | |  | | |  | | --- | | Windows is the most widely used platform, offering compatibility with Chrome, Edge, and Firefox browsers. Edge integrates deeply with Windows, but Chrome remains the most popular browser globally. Windows is user-friendly but more prone to malware compared to UNIX-based systems. |  |  | | --- | |  | | |  | | --- | | Mobile devices are predominantly used by consumers. Browsers like Chrome, Safari (iOS), and Edge (Android) dominate the mobile space. Chrome’s cross-platform integration makes it the most versatile option for both Android and iOS users, ensuring seamless access to client-side applications. |  |  | | --- | |  | |
| **Development Tools** | |  | | --- | | macOS supports a variety of development tools, including Xcode (native for iOS/macOS), IntelliJ IDEA, Eclipse, and VS Code. It excels in hybrid app development and integrates well with UNIX-based web and mobile development tools. Licensing can be a constraint for some developers. |  |  | | --- | |  | | Linux is the preferred platform for open-source development, supporting tools like Eclipse, IntelliJ IDEA, VS Code, and Docker. It offers extensive flexibility and control, making it ideal for developing and testing applications across web and mobile platforms. | Windows is widely used for development due to its comprehensive support for tools like Visual Studio, Eclipse, and VS Code. It is versatile for application and game development but requires licensing. Windows supports hybrid app development for Android using Android Studio. | Development for mobile platforms is done using Android Studio (for Android) or Xcode (for iOS). However, mobile devices themselves are not used as development platforms. Developers use macOS, Linux, or Windows for creating and deploying applications for mobile environments. |

## Recommendations

## Operating Platform The Linux platform is the most viable solution for The Gaming Room’s server-side deployment because of its flexibility, cost efficiency, and scalability. It supports high-performance operations, offers extensive compatibility with modern server technologies, and reduces operational costs. The client-side will focus on developing platform-specific user interfaces for Windows, macOS, and Android, ensuring a broad reach to diverse users while maintaining compatibility across devices.

## Operating System Architectures The client-server pattern will form the backbone of the application architecture. This approach centralizes business logic on the server side, reducing redundancy and complexity on the client side. A REST-style API will serve as the single source of truth, handling user authentication, request validation, and permission enforcement based on roles. This architecture enables independent development for client-side applications tailored to Windows, macOS, and Android while maintaining seamless functionality and scalability.

## Storage Management MySQL or PostgreSQL is recommended as the relational database for managing game data, including users, teams, and scores. These databases offer robust performance, scalability, and data integrity, which are essential for handling concurrent users and ensuring a seamless gaming experience. The database structure will be designed to efficiently store and retrieve data while supporting future application growth.

## Memory Management On the server side, memory optimization techniques will be employed to improve performance and minimize latency. This includes implementing caching mechanisms for frequently accessed data, which reduces server load and enhances response times. By leveraging Linux’s advanced memory management capabilities, the application can allocate resources effectively and ensure smooth operations, even under high user demand.

## Distributed Systems and Networks A distributed system will connect the game server and client devices using REST APIs for standard requests and WebSocket protocols for real-time communication. This setup ensures smooth team collaboration and gameplay interactions while being resilient to network outages. The platform-agnostic REST APIs support a wide range of clients, enabling developers to build device-specific user interfaces for Windows, macOS, and Android. Secure client-server communication is essential for reliability, ensuring consistent functionality across platforms.

## Security Security is paramount for protecting user information and ensuring safe gameplay across platforms. The application will use HTTPS for encrypted communication, securing data in transit between clients and the server. User authentication will rely on OAuth2 protocols, ensuring secure handling of credentials. Sensitive data, such as user information and game progress, will be encrypted both at rest and during transmission. Additionally, role-based access controls will restrict user permissions, protecting the system from unauthorized access. These measures collectively provide robust security tailored to the client-server architecture and the multi-platform nature of the game.

Citations:

Object-Oriented Programming Principles:

* MDN Web Docs. (2023, July 25). *Object-oriented programming*. Mozilla. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented\_programming

Client-Server Architecture:

* MDN Web Docs. (2016, March 1). *Client-Server overview*. Mozilla. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\_steps/Client-Server\_overview

REST APIs:

* MDN Web Docs. (2022, June 1). *REST*. Mozilla. Retrieved from https://developer.mozilla.org/en-US/docs/Glossary/REST

MySQL Documentation:

* Oracle. (n.d.). *MySQL reference manual*. Retrieved from https://dev.mysql.com/doc/

PostgreSQL Documentation:

* PostgreSQL Global Development Group. (n.d.). *PostgreSQL documentation*. Retrieved from https://www.postgresql.org/docs/

OAuth 2.0 Specifications:

* OAuth.net. (n.d.). *OAuth 2.0 specifications*. Retrieved from https://oauth.net/2/

Development Tools:

* Google Developers. (n.d.). *Android Studio overview*. Retrieved from https://developer.android.com/studio
* Apple. (n.d.). *Xcode*. Retrieved from https://developer.apple.com/xcode/
* Microsoft. (n.d.). *Visual Studio IDE*. Retrieved from https://visualstudio.microsoft.com/

Distributed Systems and Networks:

* MDN Web Docs. (2023, July 25). *Introduction to web APIs*. Mozilla. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side\_web\_APIs/Introduction

Security Considerations:

* OAuth.net. (n.d.). *OAuth 2.0 security best current practice*. Retrieved from https://oauth.net/2/oauth-best-practice/