

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 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.3 | 04/28/2024 | Derek Matias | Completed recommendations section of the documentation. |

**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)

Creative Technology Solutions (CTS) is collaborating on a project with The Gaming Room to expand the reach of their popular game "Draw It or Lose It" through the development of a web-based version. This adaptation aims to make the game accessible on multiple platforms, providing a seamless experience for users whether they are on desktop computers, tablets, or mobile devices. "Draw It or Lose It" is a drawing and guessing game where players work in teams to interpret drawings and guess the associated words or phrases. The game's transition to a web-based platform is driven by the growing demand for accessible and cross-platform gaming experiences. Our mission is to preserve the core elements of the Android version, such as the competitive spirit, the interaction among team members, and the use of a stock drawing library for clues—while enhancing the game with the capabilities of modern web technologies. The development will address specific software requirements, including support for multiple teams, unique game and team names, and the management of game instances to ensure a single, consistent gameplay experience.

## Requirements

*The web-based game application for "Draw It or Lose It" must satisfy several key requirements to ensure an engaging user experience. The game needs to support the participation of one or more teams, with each team comprised of multiple players. This feature requires a dynamic user interface and backend logic to manage teams and players effectively. To facilitate a competitive and fair gaming environment, both game and team names must be unique. This requirement necessitates a system for verifying the uniqueness of names in real-time as users select them. These unique identifiers are also to ensure the integrity of the game sessions, as it's essential that only one instance of the game exists in memory at any given time. Achieving this involves creating unique identifiers for each game, team, and player, coupled with mechanisms to manage these instances.*

## [Design Constraints](#_2et92p0)

*Developing "Draw It or Lose It" as a web-based application introduces specific design constraints that must be navigated to deliver a product that meets the client's expectations. Cross-platform compatibility is a fundamental constraint, demanding a responsive design that adapts to different devices and screen sizes while maintaining consistent performance and user experience. The real-time nature of multiplayer interactions presents another constraint, requiring robust backend support for live communication and data synchronization among players. Ensuring the uniqueness of game and team names introduces complexity to the database and backend logic, necessitating efficient algorithms for checking and managing these entities. Additionally, the requirement for only one game instance to exist in memory at any given time poses challenges for session management and server architecture. These constraints collectively guide the technical approach to developing the application, from the choice of technologies and frameworks to the architecture of the backend infrastructure and the design of the user interface.*

## [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)

1. **Inheritance**: The **Entity** class serves as a base class with common properties like **id** and **name**, which are inherited by the **Game**, **Team**, and **Player** classes. This allows for code reuse and a clear hierarchical structure.
2. **Encapsulation**: Each class encapsulates its data and operations. For example, the **Player** class encapsulates the player's details and provides a **toString()** method to represent player information as a string. This encapsulation ensures that the internal representation of each class is hidden from the outside, providing a public interface.
3. **Association**: The diagram shows association relationships where **Game** contains **Team** objects, and **Team** contains **Player** objects, indicated by the lines connecting these classes and the multiplicity (0..\*), which means a game can have zero or more teams, and a team can have zero or more players.
4. **Aggregation**: The aggregation relationship is represented by the hollow diamond shape, which suggests that **GameService** has a relationship with **Game** objects where **GameService** can contain multiple **Game** instances, but the **Game** instances are not dependent on **GameService** for their existence.
5. **Singleton Pattern**: The **GameService** class appears to implement the Singleton pattern, ensuring that only one instance of **GameService** can exist. This is deduced from the private constructor and the **getInstance()** method, which is typical for a Singleton. This pattern is useful for managing resources like a game service where a single point of access to a resource is beneficial.
6. **Use of a Service Layer**: The **GameService** acts as a service layer, providing an abstraction over the game logic and database operations. It offers methods like **addGame()**, **getGame()**, and other accessors and mutators that operate on the **Game** objects.
7. **Unique Identifier Management**: The requirement that each game and team have unique names is implied in the diagram by the **id** property in the **Entity** class. Since **id** is likely to be a unique identifier, this design ensures that each game, team, and player has a unique **id**, meeting the software requirement for unique game and team instances.
8. **Static and Dynamic Testing**: The **ProgramDriver** and **SingletonTester** classes suggest a structure for both running the application (**main** method in **ProgramDriver**) and testing its components (**testSingleton()** in **SingletonTester**), indicating an approach to static testing of the singleton behavior and dynamic testing of the program's execution.

Inheritance and encapsulation reduce redundancy and protect the integrity of the data. The singleton pattern ensures that only one game service instance is present, which aligns with the requirement that only one instance of the game exists at any time. Associations and aggregations define the relationships between the game, teams, and players, while the service layer simplifies the management and manipulation of these objects.

**"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 primarily benefit environments already invested in the Apple ecosystem. Their integration with Apple products is a strong advantage. However, they lack the widespread use and support found in other server environments, potentially limiting their applicability for a web-based application that demands broad compatibility. | Linux serves as a staple in server environments due to its stability, security, and open-source model. It allows for extensive customization to suit specific needs, supported by a large community. The drawbacks are the necessity for expertise in Unix-like systems and the occasional compatibility issues with proprietary software. | Windows servers are deeply integrated with the Microsoft ecosystem, providing seamless support for applications built on Microsoft technologies. Licensing costs and the potential resource overhead of its GUI-based tools are considerations that might detract from its suitability for certain projects. | Web applications on mobile devices must prioritize responsive design and performance. The mobile sector offers a wide audience reach, but the variability in device capabilities and operating systems introduces complexity to development and testing. |
| **Client Side** | Necessitates a commitment to aesthetic design and functionality that aligns with Mac users' expectations. Tools and IDEs specific to Mac development may incur additional costs, and the expertise required is specific to Apple's development environment. | Development benefits from a plethora of free, open-source tools, reducing software costs. However, the diversity of Linux environments increases testing complexity and requires developers to have specialized knowledge. | The large user base is a key advantage. The cost factors include potential licensing fees, and developers must accommodate a range of Windows versions. | Involves designing for a range of devices with differing screen sizes and hardware capabilities. Costs include device procurement for testing, and expertise must span both iOS and Android platforms, which have distinct design and interaction paradigms. |
| **Development Tools** | Often centered around Xcode, Apple's suite of development tools. Languages used include Swift and Objective-C for Mac-specific applications, with web development relying on standard web languages. | Varied, with Eclipse and JetBrains’ suite being popular choices. Languages commonly used are Python, Perl, Ruby, and PHP, in addition to standard front-end technologies for web development. | Include Visual Studio, with language support for C#, VB.NET, and C++. Other tools such as SQL Server are common for database work, and .NET is often used for web services. | Typically involves Android Studio for Android and Xcode for iOS. Languages include Kotlin and Java for Android, with Swift and Objective-C for iOS. Cross-platform tools like React Native and Flutter are also used to enable development across both Android and iOS from a single codebase. |

## 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**: For broadening the reach of "Draw It or Lose It" to various environments, Windows is my recommended platform. It provides a familiar environment for many developers and supports a vast ecosystem of development tools. Windows is particularly effective for organizations that are already experienced with the Microsoft ecosystem and can leverage technologies like .NET for development, which can streamline the deployment and maintenance of web-based applications.
2. **Operating Systems Architectures**: Windows operating systems utilize a hybrid kernel architecture, which combines aspects of a microkernel with those of a monolithic kernel. This architecture allows for modularity while still maintaining high performance and extensive hardware support. The hybrid nature allows Windows to run services in both user mode and kernel mode, providing a balance between performance and reliability—crucial for the demands of a real-time, multiplayer gaming application.
3. **Storage Management**: In the Windows environment, Direct Attached Storage (DAS), Storage Area Networks (SAN), or Network Attached Storage (NAS) are suitable for storage management. Windows has built-in support for storage solutions like Storage Spaces, which can be used to create flexible and resilient storage pools. For cloud-based deployments, integration with Azure Blob Storage can provide a highly scalable and secure solution.
4. **Memory Management**: Windows implements an advanced memory management system that includes features like prefetching, lazy allocation, and a large-page support system, which is beneficial for gaming applications that require quick and efficient memory access. The game could take advantage of these features to ensure smooth gameplay and quick loading times, even with many concurrent players.
5. **Distributed Systems and Networks**: For distributed systems and networking, Windows offers Windows Communication Foundation (WCF) for building service-oriented applications, which is suitable for ensuring that "Draw It or Lose It" can effectively communicate across different platforms. When considering dependencies, one must account for the Windows Server's handling of network outages, load balancing, and clustering to provide high availability and resilience.
6. **Security**: Security in a Windows environment can be managed through a combination of Windows Defender for endpoint protection, and Azure Security Center for cloud deployments. For safeguarding user information, the application should use Windows' built-in encryption features like BitLocker for at-rest data and enforce TLS for data in transit. Leveraging Active Directory for authentication and authorization provides a security framework to protect against unauthorized access to the gaming platform.