

<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 |
| --- | --- | --- | --- |
| 1.0 | Sep 20 2025 | Raynaldo Young | First Draft |

## [Executive Summary](#_sbfa50wo7nsh)

CTS has been asked by The Gaming Room to expand Draw It or Lose It from an Android app to a web version. The new design will support multiple teams and players, keep names unique, and make sure only one game is running at a time. To achieve this cleanly, we’ll use two proven design approaches: Singleton (one shared game manager so duplicates can’t start) and Iterator (a safe way to check lists to prevent name conflicts), giving players a smooth, reliable experience as the game grows. This approach will ensure a scalable, conflict free design that supports smooth gameplay as the app expands to a broader audience.

## Requirements

**Business Requirements**

* **Reach a wider audience:** Moving to the web makes the game available on more devices, bringing in more players beyond just Android users.
* **Keep the game smooth and reliable:** Names for games, teams, and players must stay unique so there are no mix-ups when people join or create groups.
* **Stay efficient:** Only one active game manager should exist at a time, preventing duplicates and wasted resources.

**Technical Requirements**

* **Support multiple teams and players:** The system must allow more than one team per game and several players per team without errors.
* **Prevent conflicts:** The software should check existing names before allowing new ones, keeping the database clean and organized.
* **Control active instances:** By using the singleton pattern, only one shared game service runs at once. Combined with the iterator pattern, the system can scan through lists safely to confirm whether a name is already taken.

## [Design Constraints](#_2et92p0)

**Single Active Game (Singleton Pattern):** Only one game manager can run at a time. This prevents duplicate games and keeps the system consistent, which is especially important in a shared web environment.

**Unique Names (Iterator Pattern):** Games, teams, and players must all have unique names. By using an iterator to check existing names, the app avoids conflicts and ensures smooth gameplay.

**Web Deployment Compatibility:** The system must expand from Android to the web. This means the design must connect to the existing backend while also working in browsers, without breaking current features.

## [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 UML diagram shows how the classes in the application work together to meet the client’s needs. The **ProgramDriver** class is the starting point of the program. It calls the **GameService**, which manages the creation of all games. **GameService** is built using the Singleton pattern, which means only one instance of this service can ever exist. This is done by making its constructor private and only allowing access through the **getInstance()** method. If the instance does not exist, it is created; otherwise, the same instance is reused. This ensures that multiple versions of the game manager cannot run at the same time.

Once **GameService** is running, the driver can call the **addGame()** method. This method uses the Iterator pattern to scan the list of existing games and make sure that no game with the same name already exists. If the name is unique, a new Game object is created and added to the list. The same logic applies when a Team is added to a Game, and when a Player is added to a Team iterators are used to prevent duplicate names and keep everything organized.

The Entity class is a base class that holds shared attributes (id and name) for Game, Team, and Player. By placing these in a parent class, the design uses inheritance to reduce duplicate code. This also demonstrates abstraction, since the details of how IDs and names are managed are hidden from the user, who only interacts with higher level methods like **addTeam()** or **addPlayer()**.

Other key object-oriented principles are also clear in the diagram. Encapsulation is shown because attributes like id and name are kept private or protected and can only be accessed through methods. Polymorphism appears through constructor overloading and shared behaviors across subclasses. Together, these principles create a design that is reusable, easy to maintain, and aligned with the software requirements.

Overall, the domain model ensures that only one game manager exists at a time, names are kept unique, and relationships between games, teams, and players are managed cleanly. The Singleton and Iterator patterns support these goals, while core object-oriented principles make the system efficient and scalable.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macs are Unix-based and stable, but they are expensive and not typically used as servers. They’re better suited for development workstations rather than production hosting. | Linux is the standard for web servers because it’s open-source, secure, and very reliable. It’s widely supported and cost effective, making it the best fit for hosting. | Windows servers are dependable and integrate well with Microsoft tools, but licensing fees make them more costly. They’re common in corporate environments but less popular for hosting web apps compared to Linux. | Mobile devices aren’t designed for hosting applications. Instead, they connect to servers, so they serve as clients rather than back-end hosts. |
| **Client Side** | Mac users expect apps to run smoothly in Safari, which can require extra testing. Supporting Mac is important but usually adds cost and testing time since the user base is smaller. | Linux users often rely on Firefox or Chrome. It’s less common for general users but popular with developers, so testing should include Linux browsers even though it’s low cost. | Windows is the most widely used client platform. Supporting it is essential, but it requires more testing across versions and browsers to make sure everything works consistently. | Mobile is critical because most people access apps on their phones. Both iOS and Android must be supported, which increases development time and budget. |
| **Development Tools** | Macs support Xcode (needed for iOS apps), plus IntelliJ and VS Code. They handle Java, Swift, and JavaScript well, so they’re useful for cross-platform development. | Linux offers powerful open-source tools like Eclipse and VS Code. It’s excellent for Java, Python, and PHP, which are common for web apps. | Windows supports Visual Studio and Eclipse, making it strong for .NET, C#, and Java. It’s also highly compatible with enterprise software. | iOS apps require Xcode on Mac, while Android apps use Android Studio. Cross-platform tools like Flutter or React Native help reduce time and cost by supporting both platforms at once. |

## Recommendations

1. **Operating Platform**: Windows is the best fit for expanding Draw It or Lose It. It’s the most widely used system, supports a huge range of software tools, and makes development and deployment straightforward.
2. **Operating Systems Architectures**: The Windows environment allows applications, files, and network services to work together without extra setup. This makes it easier to build, test, and run the game in a consistent way.
3. **Storage Management**: Windows 10 and 11 include Storage Sense, a tool that automatically clears temporary and unused files. This helps free up space and keep performance steady as game data grows.
4. **Memory Management**: Windows uses a paging system and task manager to allocate memory where it’s needed most. This reduces crashes and ensures the game runs smoothly even when other apps are open. This means smoother gameplay and fewer interruptions for users.
5. **Distributed Systems and Networks**: By using a central database and reliable server connections, Windows can handle player logins, team data, and communication across different devices. This ensures everyone stays connected in real time.
6. **Security**: Windows includes strong security features like Defender and built-in firewalls. Combined with regular updates and optional third-party tools, it provides protection against malware and unauthorized access, it is a safe platform for storing and protecting user information.