

<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 | 05-22-2025 | Julliane Pamfilo | Initial draft completed. Added executive summary, requirements, design constraints, architecture view, domain model, evaluation, and recommendations for Draw It or Lose It. |

**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)

The Gaming Room has asked Creative Technology Solutions (CTS) to enhance and extend their current Android game, Draw It or Lose It into a web enabled application aiming to meet the demands of its ever-growing user community. It is based on an energetic, no-holds bar version of the game, teams guess fast and advance their token on the game board to a Victory space. It’s the gameshow IRL.ACLE connects to multiple devices and multiple players teams. With the gameshow mode, one device is the host and players connect to the host using their devices. Bluetooth access is also required.

It is essential to come up with an architecture that caters to the client requirements and for this purpose in this software design, a modular object oriented driven architecture is proposed that incorporates fundamental principles such as inheritance, encapsulation and reusability. One of the central elements is a Singleton-based GameService class, which makes sure there is always only a single instance of the gameplay logic loaded into RAM. It ensures that the recreation of the state still occurs in only one place and will obey the same lifecycle in multiple sessions. You could also implement the Iterator pattern and make sure game and team names are imported once by checking after-the-fact if there are duplicates.

This design is the cornerstone for building the scalable distributed application that works across a set of platforms, provides client’s core game play logic, and abstracts future hardware and network development.

## Requirements

The Gaming Room needs a simple 1-level, browser-based version of their Android game Draw It or Lose It. The main requirements in software are as follows:

* The system would need to be able to accommodate multienrollment with multiple players per team.
* No two games, teams should have the same name and user should be able to verify the availability of the name.
* The game application needs to have just one instance running in memory during execution, so we apply the Singleton pattern.
* A game, team and player need an unique identification to be able to handle and monitor them in efficient manner.

These needs are important if you want scalable and device-independent gaming.

## [Design Constraints](#_2et92p0)

Porting Draw It or Lose It as a web-based distributed application comes with several important limitations that need to be considered in order to achieve efficient and reliable performance across browsers.

There is one major limitation, that all runs are finished at the same time, so this is fixed with Singleton design pattern. Such a restriction serves to maintain game state consistency and guarantees that not more than one conflicting game plays at the same time.

A further restriction is the prohibition of double names in the game and the team. This is absolutely vital for user-friendliness and data integrity, and it's done automatically by the Iterator pattern to ensure that no duplicates made before new entities are created.

Moreover, because the application is web-based and distributed, performance optimization, scaling and resource consumptions should be considered. Performance is crucial here because of the real-time drawing, so there must be a smooth rendering with a predictable rate for each round.

Finally, the application needs to be designed as modular and reusable, for allowing the platform future growth and the connection to front-end interfaces, and eventual hardware choices built on this software architecture.

## [System Architecture View](#_ilbxbyevv6b6)

The architecture of Draw It or Lose It operates under a three-tier architecture:

* Presentation Layer (Client Side): Website or later mobile Frontend which is responsible for the communication with Player.
* Application Layer (Server Side) – This is where the Java logic resides on the server. Includes logic on creating games, managing teams and receiving players from GameService Singleton.
* Data Layer – Not yet implemented, future iterations could introduce persistent storage and/or database for saving game sessions, players and teams.

The is application is modularize and ready to be used distributed. The myrmidon lays out concerns strategically, separated from the logic, so that bricks can be deployed across servers and platforms, enhancing the scalability of the deployment in further release cycles.

## [Domain Model](#_8h2ehzxfam4o)

The UML Class Diagram for Draw It or Lose It shown on Designed on object-oriented principles, it makes use of inheritance, composition and design patterns.

At the top of this is the Entity base class, that has 2 base attributes id (a unique identifier) and name. This class implements constructors and getter methods and is extended by 3 other classes Game, Team, and Player. This is the way to keep away from duplication and scalability.

GameService: This is the main service that manages all game objects. It holds collections of games and has methods to save and load them. This class is designed as a Singleton in order to guarantee that there is just one instance at a given time (and this is what we need – just one live game at once).

Each Game object has a list of Team objects, and each Team has a list of Player objects — composition at work! Methods addGame, addTeam and addPlayer have the logic of ensuring unique names (shared unique instance by the pool).

The ProgramDriver class sets up the program by communicating with GameService and the SingletonTester class checks that Singleton is working properly.

**"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 a Unix-like environment for development and testing locally but not in large-scale hosted environments. Good security and performance, but lacks native support for several Java enterprise frameworks and perhaps incurs higher hardware prices. | If the operating system on your server is Linux, your solution is server installation is great, Linux is a stable scale able to Open-source cost effective 24/7 support. It has very good support for Java servers out of the box, it is popular in cloud deployments and has both active community and enterprise support. | If you prefer Windows Server is a possibility but will use more resources and be more expensive with the licensing cost. While Java is possible, the platform is predominantly used for. NET stacks and may need further configuration for Java environments. | Mobile phones aren't capable of a server hosting. Instead, they phase in as clients, which are therefore clients to the server-hosted application. That has to take place server side in a cloud or on premise estate. |
| **Client Side** | Works with newer browsers such as Chrome and Safari. Website owners have to test reaction time and appearance in Safari in particular. Apple hardware is required, leading to costly investment in development. In general, modern web standards are bringing to the Mac enough support on the client side. | If user is on the desktop, then user probably uses Firefox or Chrome with Linux. It works fine with contemporary browser standards so little can go wrong there. But there are subtle differences between the various distros that can hurt UI rendering, and thus it has to be tested more widely. | A popular platform, with Chrome and Edge as dominant browsers. UI and interactions must be meticulously tested for Windows desktop to achieve the same accessibility and performance. | Needs a mobile-first-web-based UI which is compatible with iOS Safari and Android Chrome. Developers need to consider smaller screens, varying touch inputs and lack of them, slow speeds and browser differences. For chasing down bugs, in particular, emulators and mobile debug tools are must-haves. |
| **Development Tools** | You can develop in Java from any IDE or other editor (like VS Code) that you want, in macOS. Support for terminal development is solid, and you have things like Git built in. Apple’s licensing costs and hardware needs can add to team costs. | Linux is very well suited for Java development. It provides full toolchain integration (eg, Eclipse, IntelliJ, NetBeans), strong command-line tools, and support for deployment pipelines. It also serves for backend, because it is cheap. | Windows does also support Java development through IDEs such as IntelliJ IDEA and Eclipse. With PowerShell, Git Bash & easy JDK installation, development is possible for everyone. But there are some open-source libraries that once in a while there is something that doesn't work on it. | Development doesn’t happen in the mobile device themselves, but testing does require some cross-platform debugging tools such as Chrome DevTools and Safari Web Inspector. Simulators (e.g., Android Studio, Xcode) are needed for the simulated testing mobile interaction and performance. |

## 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 cross-platform support, scalability and open-source versatility, Linux should be your choice of operating system. It has proven popular in server environments as it natively supports Java-based applications and achieves great performance with very little overhead.

1. **Operating Systems Architectures**:

* The system will be implemented on a 64-bit Linux OS with a modular, service-oriented design. This in turn enables memory efficiency for large-scale distributed processing, as well as higher performance for the rendering of real-time game data and the mutual interaction game players.

1. **Storage Management**:

* Whereas the real-world equivalent on which it is based is played with real playing cards and directors’ roles are taken by one or more of the players, no effort is made to store persistently game sessions, data on individual players and the histories of one or more teams or the results of individual tournaments, although later such a system could optionally be coupled with some kind of relational database server (played in sequence or in parallel with an R Db such as PostGres or MySQL). This storage was intended to be available through a RESTful backend which the GameService logic was attached to.

1. **Memory Management**:

* Memory management in JVM: In other words, Memory management by garbage collection and dynamic allocation by the Java Virtual Machine (JVM) is performed. The Singleton pattern above makes sure we only create one GameService instance, minimizing memory usage and avoiding memory leaks.

1. **Distributed Systems and Networks**:

* The backend will have to be hosted to a distributed system, e.g. the cloud, to support cross-device usage. Client requests from web browsers or mobile devices can use RESTful APIs so the server can maintain state between requests. Dependencies includes the availability of the network, fail-over management, as well as session management.

1. **Security**:

* Secure client-server connections should be established, albeit in order to keep user data as well as gameplay sessions safe and sound. Also, future versions should take the advantage of the authentication (for example: OAuth2) and secure storage of information while encoding user IDs. There's solid firewall and security tools in the Linux OS though.