

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 |
| --- | --- | --- | --- |
| 1.0 | 05/26/2024 | Pawan Gaire | First Project |
| 1.1 | 06/08/2024 | Pawan Gaire | Second Project (Evaluation) |
| 1.2 | 06/22/2024 | Pawan Gaire | Project Three (With Recommendations) |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room requested CTS to create a web-based version of their Android game, Draw It or Lose It. The new version should support multiple teams, each with various players, ensuring that each game instance, team, and player is unique. To achieve this, a singleton pattern is used to create game instances to avoid duplicates, and an iterator pattern is implemented to manage teams and players, preventing conflicts.

## [Design Constraints](#_2et92p0)

The Gaming Room already has an Android version of Draw It or Lose It. CTS needs to create a web version, so they picked Java to do it because Java is already used for Android, making the process easier. They also need to check and possibly update any APIs used for the Android version to make sure they work well for mobile use.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram for the suggested design is displayed below.

**"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 application has a main driver class that starts the process of creating games, teams, and players. The GameService class handles the actual creation and uses a singleton design, meaning only one instance of GameService can exist at a time. To prevent multiple instances, GameService has a private constructor and can only be created using the getInstance() method, which checks if an instance already exists before creating one. Once GameService is running, the driver class can create new games using the addGame() method. This method ensures no duplicate game names and adds the new game to a games list. After a game is created, we can add a team using the addTeam() method, which prevents duplicate team names and adds the new team to a teams list. Similarly, we can add a player to a team using the addPlayer() method, which prevents duplicate player names and adds the new player to a players list.

Game, Team, and Player are all subclasses of the Entity class, which has protected id and name attributes. Only specific constructors can be used to prevent null objects. The design shows object-oriented techniques like polymorphism and inheritance in the Entity class, and encapsulation and abstraction in the addTeam() method. Teams can't be created directly; instead, they are added using the addTeam() method without the user needing to know the details.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | One big plus of MacOS is that it can run apps from MacOS, Windows, and Linux all at once. Another perk that people don't always notice is how consistent MacOS is. It stays pretty much the same over time, so if you're used to it, it's easy to use. But MacOS can be expensive, and you have fewer hardware choices compared to Windows or Linux. | Linux has many distribution options, and it's free and open source with lots of inexpensive versions available. It's famous for being flexible, especially for servers and embedded systems, because you can customize it easily. Plus, it's generally more secure than Windows or MacOS. Downsides include fewer options for buying pre-made computers, and sometimes files don't work well across different systems. | Windows is great for businesses because it works smoothly with their corporate servers like Active Directory without needing extra costs. But it falls short in mobile app development and its security isn't top-notch, leaving it vulnerable to viruses and cyber-attacks like malware or ransomware. | Using a mobile device to run a web application can be useful if there aren't many users and the app isn't too big or complex. Android users have access to inexpensive web server apps, which is a plus. However, the downside is that mobile hosting options are often cloud-based, which can make companies more vulnerable to hackers. Plus, mobile devices are generally less secure than PCs. |
| **Client Side** | MacOS is great once you know how to use it, but it's only available on Apple devices. This means if you're not using an Apple device, you can't access it. This limitation might be frustrating for developers who know how to make MacOS programs but don't have a Mac to work on. | Linux has its advantages in terms of cost-effectiveness and the freedom it gives you in developing software. Since Linux is free and open source, it's usually easier to maintain and requires less time for upkeep. However, there are drawbacks to these benefits. Because Linux is open-source and user-controlled, security might become a concern, especially when compared to the technical support provided for Windows or MacOS. | Windows has its advantages, like being widely available and offering a range of prices that can be adjusted to fit your project's needs. They provide technical support and better security options compared to Linux. However, you'll need someone who knows their way around Windows, and there might be extra costs for certain features you want to add. | Although there are numerous apps and tools for mobile devices, they often don't have the same accessibility and complete features as those on a PC. The upside is that they're widely accessible and come in various price ranges. However, the downside is that mobile devices run on different operating systems, and each device is typically designed for a specific OS, which can make them incompatible with other systems. |
| **Development Tools** | MacOS uses Swift for programming. The main tool for MacOS and iOS developers is Xcode, along with Xcode Cloud. Xcode Cloud is a service made for Apple developers, making it quicker and easier for teams to create, test, and release apps efficiently. | Linux offers a ton of development tools, so let's focus on one: Docker. Docker helps by giving a consistent environment for development, making it easier to build apps that work on different platforms, and deploying them smoothly. Docker Hub is also handy because it lets users skip setting up the development environment and jump straight into coding. | Windows was mainly made using C, with a bit of assembly language here and there. Visual Studio is the most famous and widely used tool for working with Windows. It's not just an Integrated Development Environment (IDE), but also a code editor and sometimes even a tool for controlling source code versions. Visual Studio offers a bunch of useful features. Although I was a bit slow to realize its advantages, now I mainly use it for both school and work projects. | Java is a popular choice for making mobile apps because of its object-oriented features. Developers often use it because it's well-suited for creating mobile apps. But other languages like Python and C++ are also used, especially for making games. There are many tools (IDEs) for making mobile apps, but some of the most popular ones are VSCode, IntelliJ IDEA, and Eclipse. I don't like Eclipse because it feels cluttered. Instead, I prefer using Visual Studio with Xamarin, which lets me make apps that work on different platforms. |

**Recommendations:**

1. **Operating Platform**: Use Linux-based servers for the backend to save on licensing costs and gain better access to data centers. The frontend can connect to the backend through APIs, making it possible to develop in languages like SWIFT for iOS, Java for Android, or .NET for Windows. Linux is secure, reliable, and has many available tools, including security software.

2. **System Architecture**: Use a backend server for game logic and a client-side frontend for rendering. Since Draw It or Lose It doesn't need low latency, asynchronous data transmission is fine. A scalable backend with containerized microservices using Kubernetes or Docker is recommended. Choose a cloud provider first, as their tools will affect the architecture. Client-side rendering reduces server load and costs, ensuring smooth gameplay by preloading images. Decide if the game should be browser-based or a Java app for PCs/Macs, with browser-based and PWA options being simpler.

3. **Storage Management**: If we are not buying proprietary hardware, there's no need to choose between HDD and SSD, as both will work well with caching and client-side rendering. Using cloud-native tools for storage adds flexibility and scalability.

4. **Memory Management**: Use Linux for efficient memory management with page cache and demand paging, which loads only necessary pages. Android’s ART and Dalvik VM keep modified memory in RAM, while iOS uses Automatic Reference Counting (ARC) managed by Xcode. Minimal server RAM is needed due to client-side rendering, with cost scaling based on user numbers. Client-side RAM needs are low, storing only a few images and the client application.

5. **Distributed Systems and Networks**: Use cloud-native architectures to improve uptime and prevent outages, with cloud providers able to replicate and shift services as needed. Ensure frontend and backend communication through RESTful APIs asynchronously, working seamlessly across platforms like Android, Windows, and iOS.

6. **Security**: Implement role-based authorization to control user access, limiting permissions to necessary game functions. Ensure no users have ADMIN rights. Protect APIs with SHA-256 encryption, 128-bit keys, and at least TLS 1.2. Get certificates from Entrust. Set up a firewall using industry-standard best practices to secure the server.