

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | August 22, 2025 | Grace Wright | Completed Recommendation. |
| 2.0 | August 9, 2025 | Grace Wright | Completed Evaluation. |
| 1.0 | July 21, 2025 | Grace Wright | Completed Executive Summary, Requirements, Design Constraints, and Domain Model. |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is a co-op game available only as an Android application. Client The Gaming Room wishes to develop a web-based version of the game that will function for Windows, Mac, iOS, and Android operating systems. Games should not run concurrently in memory, and teams and players should be unique.

## Requirements

* Hosting games will require client-server architecture.
* Games can exist only one-at-a-time in memory.
* Access across all popular devices and browsers is required.

## [Design Constraints](#_2et92p0)

* Games, teams, and players must have unique identifiers so as to prevent duplications. A Singleton design pattern may be employed to prevent repeat instantiation of entities.
* The web game must be compatible with major browsers: Chrome, Firefox, Safari, and Edge.
* Both desktop and mobile versions of the website will need to be developed for each browser.
* Game names must be unique; Team names must be unique; all Entities should bear unique identifiers; one or many teams may play one game; one to many players may exist on each team.

## [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 **Entity** class serves as the parent class to **Game**, **Team**, and **Player**. **Entity** is a general form of object for the purposes of the game’s development; it provides the three inheriting classes with basic attributes that are common to all objects, namely **name** and **id** with related accessor methods and constructor. Each inheriting class then builds upon this base with its unique attributes, encapsulating them from the other inheriting classes: i.e., the unique **toString()** function present in each child class may behave differently—a trait known as polymorphism.

The lines between classes in the UML diagram denote the cardinality that was outlined under **Design Constraints** above; *e.g.*, the “0…\*” between **Game** and **Team** implies that there may be zero (“0”) games but that multiple (“\*”) instances of **Team** may exist under (“…”) an instance of **Game.**

Disconnected diagrammatically from the **GameService** and **Entity** classes are the **ProgramDriver** and **SingletonTester**. In conjunction, these classes drive the main program: code will begin to execute from an entry point in the **ProgramDriver** class file (from the **main()** method). This class utilizes the **SingletonTester** class to maintain the one-game-only rule based on the singleton design pattern mentioned under **Design Constraints** above.

**"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** | | | | |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | A Snow Leopard Mac OS X Server costs $499 with the Server app costing $50. However, Mac discontinued support for Mac OS X in 2022; while still usable, it is no longer supported with updates or security fixes. Server side functions are now integrated into all macOS, meaning startup should be streamlined. Server hardware can cost upwards of $6,000. | Maintaining and licensing Linux webhosting OS is cheaper than Mac or Windows. Linux also receives preferential treatment by major cloud providers like Google. | Windows is ubiquitous and users and developers alike are accustomed to the OS.  Windows servers have high licensing costs. The datacenter edition’s current suggested MSRP is $6,771. | Mobile devices are not fit to serve as servers for serving multiple other clients. The application should be designed to function on mobile devices (client side), but serving using mobile devices is unrealistic for this project. |
| **Client Side** | Upfronts costs are low as the HTML-based web application can be handled easily by the default Mac browser Safari. | Linux systems will likely be loaded with a preferred browser by the users. Accessing the web-based game is a simple matter of browser compatibility. | Windows is no different from Mac OS in that a browser will be used to access and handle the web-based game. | The application will need to be reworked for iOS as an inbuilt application.  Users will not enjoy having to utilize the Safari app to navigate to the website and play an embedded version of the game via the HTML interface, which may have rendering issues. It is possible to develop the website’s mobile version to play well with the Safari app, but developing a new application for the Apple Store should be preferential. |
| **Development Tools** | Mac OS development languages include C and SWIFT.  Developers commonly prefer Xcode, which costs $99/year for each developer. | Linux developers utilize C, C++, Java, and Python.  PyCharm is a popular IDE for Python users.  Eclipse can handle all of the above languages and is free to developers. | Windows developers use C# and .NET.  Visual Studio is a comprehensive IDE for Windows users but can cost up to $100/month for the Professional Standard option. | Android developers utilize Java.  Android Studio is the preferred IDE and is distributed freely by Google. |

## 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**: Frontend development will take place on device-native operating systems at the clientside. The frontend-to-backend connection is preserved regardless of operating system chosen for the servers.

On the backend or server side, Linux is recommended. Many developers are familiar with and prefer Linux operating systems. Linux, being open source, dodges high commercial licensing costs associated with Windows and iOS.

1. **Operating Systems Architectures**: Client-server architecture allows the Linux servers to communicate with Android, iOS, Windows, and Linux clients. Note that Windows and some iOS clients may be accessing via the web-based version of the game.
2. **Storage Management**: At the clientside, users will store game data or cookies, depending on whether they download an app or use the website, locally on their machines.

As for the servers, cloud-based server management is preferred. Google Cloud Services are recommended as they are industry leader in hosting servers for the more intangible business operations like hosting multiplayer online games. Cloud storage is preferred because it is difficult to determine the exact amount of storage in-house physical servers could need, and mistakes can be costly. Utilization of cloud services will eliminate the uncertainty and potential high costs.

1. **Memory Management**: Memory management is not a concern at the clientside, as each game necessitates holding only a single image in memory while it is rendered during gameplay.

On the servers’ side, the burden is similarly minimal as the rendering occurs on the clients’ machines after the server has transmitted the image data, which is its significant memory burden at any given time. The memory available (and used) via the chosen cloud service will be adequate and cost-effective.

1. **Distributed Systems and Networks**: As discussed in Recommendations list item #3, cloud management means lower uncertainty and paying only for what the company uses. Client-server architectures have other advantages: cloud service providers can reroute computing operations to other loci in their servers when outages occur, preserving business operations.

RESTful APIs should be utilized for client-server interactions as it will bolster security and prevent communication errors that could affect game performance.

1. **Security**: Several operations to improve security should be taken:

Least privilege: User permissions should be limited to only what each absolutely needs for their purposes. Unique roles in the system will allow differentiations to be made between admins and users.

Encryption: Sensitive data will not be sent in plain-text format; nor will passwords be stored this way.

Logging: Records should be generated and stored to track or review suspicious activity.

Authorization: The game is not privy to sensitive user data with the exception of perhaps user password choice, which should be encrypted in the first place. Thus, two-factor authentication may make the user experience cumbersome. One-factor authentication via a username and password system should be adequate for verifying user logins. Multiple attempts within a short time period should trigger blocking, or locking the user out of subsequent attempts for a time.