

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 |
| --- | --- | --- | --- |
| 1.2 | 06/18/23 | James Furman | Wrote recommendations section |
| 1.1 | 06/04/23 | James Furman | Wrote evaluation section |
| 1.0 | 05/21/23 | James Furman | Wrote executive summary, requirements, design constraints, and domain model description. |

**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 needs a web-based platform on which to run their game, Draw It or Lose It. The platform must allow multiple players to join, on multiple teams, in multiple games. Hardware requirements will be determined in the future based on software development decisions.

## Requirements (B: business requirement; T: technical requirement)

(T) A game can involve one or more teams

(T) Each team will be assigned multiple players

(T) Game and team names must be unique

(T) Users must be able to check whether a name is in use when choosing a team name

(T) Each game, team, and player must have a unique identifier

(T) No more than one instance of the game can exist in memory at a given time

(B) The Gaming Room’s requirements for building the game on this environment will be determined by the environment’s implementation

## [Design Constraints](#_2et92p0)

* The GameService class must use the singleton design pattern.
* The Game, Team, and Player classes must use iterators to enforce unique identifiers and names.
* ID generation will be handled automatically.
* Players must be able to input names and receive feedback if the name input is already in use.
* The Game, Team, and Player classes must implement an abstract Entity class.
* The GameService class must contain a method for adding Game objects, the Game class for adding Team objects, and the Team class for adding Player objects.
* Enforcement of multiple players per team will not be handled in the object model (implemented by, e.g., preventing a game from starting if a team has less than 2 players).

## [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 contains a long-type id and a String name and has constructors and accessor methods. The Game, Team, and Player classes extend the Entity class. The Game class has a method to create Team objects, and the Team class has a method to create Player objects.
* The singleton GameService contains the methods for adding new Game objects and for generating and storing ids for Game, Team, and Player objects. It may include zero to many Game objects, each Game object may include zero to many Team objects, and each Team object may include zero to many Player objects.
* The ProgramDriver class contains main() and uses the SingletonTester class, which contains a method to test the implementation of the singleton pattern for GameService.

**"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 can run a web server with MAMP. MAMP is a free stack, but macOS can only be run on Apple hardware, which is costly. The OS is reliable and stable, therefore scalable, and has native security features. | Linux is open source and can be run on any hardware. It is stable and reliable for scaling. It is designed securely, and its open-source codebase makes it fully open for scrutiny to ensure security. XAMPP is a free stack for running a web server on Linux (also macOS & Windows). Linux has a higher learning curve, and its distributions and software are often fragmented, so developers must be experienced. | Windows has the Microsoft IIS solution available for Windows Server, which includes native Windows security features. For the scale of The Gaming Room’s deployment, we should expect a $1069 (MSRP) Windows Server licensing fee, but the cost can go as high as $6155. Windows can also use MAMP or XAMPP. | Mobile devices have the advantage of portability. However, their computing resources are limited compared to the hardware used for desktop operating systems, they rely on relatively unstable wireless internet connections, and they have fewer available security tools. |
| **Client Side** | Because Draw It or Lose It is an HTML webapp, its client-side experience does not differ on desktop OSes. Any Mac-compatible, up-to-date browser can load Draw It or Lose It. Apple computers use slightly different keyboards, so hotkeys may need to be reconfigured for browsers on macOS. | Any Linux-compatible, up-to-date browser will be able to load Draw It or Lose It. | Any Windows-compatible, up-to-date browser will be able to load Draw It or Lose It. | Draw It or Lose It is exclusively online, so no app needs to be developed for offline play (e.g. a Progressive Web App). Apple’s WebKit (FOSS) can be used to develop a responsive HTML browser for both iOS and Android. For mobile loading, the HTML generated by the server must be responsive. The touch interface may produce bugs, and may give players unfair advantages or handicaps, so developers must take UI and game design into account. |
| **Development Tools** | MAMP – Apache or Nginx server, MySQL & PHP.  Eclipse IDE, Java, Maven build tool, dropwizard framework for REST API.  All free.  macOS has a user-friendly dev environment, but customizability is limited. | XAMPP – Apache server, MariaDB, PHP, Perl.  Eclipse IDE, Java, Maven build tool, dropwizard framework for REST API.  All free.  Linux has a vast software ecosystem that can present a learning curve of its own, but familiar devs may find it a more favorable platform for development. | MAMP – Apache or Nginx server, MySQL & PHP.  Eclipse IDE, Java, Maven build tool, dropwizard framework for REST API.  All free.  Windows has a wide base of native software that devs may find useful. | WebKit API in C++ to develop the app’s browser.  For server dev (bad idea), develop for Linux deployment and use iSH on iOS or Termux on Android to emulate Linux.  All free.  Devs will need to be familiar with WebKit API, and the development itself must be done on a desktop platform. |

## 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**: I recommend The Gaming Room use a Linux server to run their server-side application. macOS requires relatively expensive hardware and Windows confers no particular benefit over Linux, and while implementation on Linux involves a somewhat higher learning curve than Windows’s proprietary and costly solutions, there is no shortage of developers capable of handling it. Ubuntu is a popular, well-maintained, and secure open-source distribution of Linux and an easy choice for deploying the server.
2. **Operating Systems Architectures**: The server will be built with XAMPP. It will use a MariaDB database, which is based on MySQL and highly compatible with MySQL code. It will run an Apache HTTP server app that supports REST. With the JVM installed on the server, the game platform application, packaged as a jar file, can be called remotely via a bash script; this application uses dropwizard to parse Java objects into REST calls which use Apache to communicate with the client application over the internet. Server utility functions should (probably) be written in PHP or Perl for more direct integration in XAMPP.
3. **Storage Management**: Storage should be done on a random-access disk drive. Since not much storage space is required for Draw It or Lose It’s requirements, The Gaming Room can inexpensively purchase hard drives for their server hardware with sufficient space to store the server application, user data for authentication, and the game’s image library. These drives can also have extra room in case the library grows as well as potentially providing space for useful files like records of logins (to analyze usage for development or marketing purposes).
4. **Memory Management**: Draw It or Lose It’s server applications will be perpetually loaded in main memory on its server. Since the server is dedicated, it can reasonably have enough memory to handle service to a sufficient number of clients for Draw It or Lose It’s initial requirements; however, if the game meets with success, memory limitations may become a practical constraint on the number of players who can be served concurrently. It may instead be desirable or even necessary, for memory management’s sake, to deload connections to the platform after the game has been initialized. Implementing this approach may be a UX compromise given the possibility that players will desire to join Game rooms after beginning play; forcing a group of players to begin a new session is less desirable than allowing dynamic joining and quitting.
5. **Distributed Systems and Networks**: Draw It or Lose It will communicate between devices via the internet, meaning that the client application needs to be hosted online but not necessarily on a distributed system. However, it can be hosted on a serverless platform such as AWS, which could provide faster service to users connecting across a broad geographical area. Serverless hosting would also provide greater redundancy against service loss, but would make The Gaming Room entirely reliant on AWS’s undisrupted service and its terms of use, meaning that AWS would itself become a single point of failure; which option is preferable may be more a philosophical question than technical.

It would be preferable to host the game’s image library on a distributed network so that when clients connect to the server application, the server can open a full-duplex pipe between the client and the image database and the client application can send calls to the image database to download images to the client’s browser (or mobile app) without needing to access The Game Room’s on-site server.

1. **Security**: User login information, as well as any other personal or sensitive information provided to either the server or client application, should be encrypted before being sent remotely, and kept encrypted in the server database. Authentication and authorization must be applied to every system connecting to the server, developers must be careful to implement these measures correctly, and they should be rigorously tested before deployment. The Game Room should make use of the security features available for Linux servers, and both the client and server applications should be developed securely and tested for security to ensure exploits are not exposed.