

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_gjdgxs)

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/2023 | Chris Ellis | Initial Draft |
| 1.1 | 11/25/2023 | Chris Ellis | Add Evaluation Section |
| 1.2 | 12/10/2023 | Chris Ellis | Add Recommendations Section |

**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](#_35nkun2)

CTS’s new client The Gaming Room has come to us asking for help taking their existing game Draw It or Lose It, from being available only on Android to a web-based game that can be accessed from multiple platforms. The game itself involves rendering drawings over the course of 30 seconds while teams attempt to guess what the picture is before the time expires, if the pictures hasn’t been guessed, other teams have a 15 seconds window to enter one guess of what it might be.

The Gaming Room has also specified very specific requirements for how the game platform should operate in regards to naming of teams, and games, along with how game instances are managed by only one instance of the game service. They have also expressed concern with their ability to set up the environment, and would like us to create a software design document to facilitate planning for the application, along with starting to develop the game application.

In response we’ve created this software document to cover the design portion, with plans to begin programming the game service, which will be written in Java, a multi-platform language.

## Requirements

(As specified by the client in project guidelines)

1. A game will have the ability to have one or more teams involved.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
4. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_1ksv4uv)

There are multiple design constraints involved in porting this game from a single mobile platform to a web based platform.

1. Client Server Model
2. Multiple Client Operating Systems
3. Team Composition related to developing for multiple platforms
4. Design pattern choices for Game, Team, Player entity and unique id requirements

**Rationale**

1. The game was originally developed for the Android OS, and moving to a web based platform in order to meet the requirement of serving clients on different devices will require using a client-server model. Java is an excellent choice for a programming language for the server as it does not need recompilation to run on multiple platforms, allowing for multiple choices for the server side OS.
2. This is a balancing act between the cost of having multiple languages used (polyglot), where certain tools will be easier to use on their respective platforms, versus having a single unified language used across the board even if some pieces may be more complex with that method. In addition native mobile client apps generally provide a better interface than the browser on a mobile platform due to the tight integration of the previously mentioned tools and libraries with the hardware and operating system of their respective platforms.
3. Team composition will need to be balanced based on the choices of programming languages, as Java can also be used to program the client applications, being it can run on multiple platforms. There may be a better option in using Java for PC clients and using mobile device’s preferred programming environments like Android studio and XCode with Swift or Objective-C on iOS. This obviously will require a better
4. The requirements from The Gaming Room for having multiple teams and players in a game, along with multiple games inside a single game service, and that game service being the only instance running at one time, lead to needing to apply specific design patterns to achieve these goals, like a Singleton game service along with specific entity multiplicities for the game, team, and player entities that will need to exist.

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

The below UML class diagram represents the initial design of the game service application. We can start by looking at how the program will operate and drill down into the features that this design offers.

We can see the ProgramDriver class which is the main entry point to start the program and manage the state of games contained within the GameService. The ProgramDriver uses the SingletonTester class’s testSingleton method to perform a test to ensure we are meeting one of the major requirements The Gaming Room has given us for the product, which is that there can only be one instance of the GameService in memory at a time. This was achieved by using the OOP pattern of Singleton to ensure that only one instance is created, by checking if there is an instance in existence, and if so returning that instance.

Diving into the GameService class, which is the main object controller for the application, we can see that it possesses a list of current games, with the ability to add games, and get info about games it currently manages. Following the object association lines we can see by the marked multiplicities that a game service can map to many games, a Game can have many Teams, and a Team can have many Players, Games/Teams/Players are Seperate classes each with their own lists of the sub-objects with Player’s not having a list as it is the lowest object in the hierarchy. This covers the first two requirements given to us, as a Game can have multiple teams, while a team can have multiple players. As these classes all have common properties, being a name and an id, we are extracting those properties into a parent class that all of these can inherit from, called Entity.

There are many other OOP programming principles shown in this diagram that will assist us in creating clean code that is modular and reusable. Abstraction is shown in multiple places, the most obvious being the direct inheritance of ‘Entity’ class by its child classes.. As id and name are private in the parent class, and there are no setter methods, these properties can only be set when calling the constructor of the child class, ensuring that the data is not mutable, which is an example of the OOP principle of encapsulation.. This helps us meet another hard requirement we were given which is that players on a team should not be able to have the same name, and teams in a game should not be able to share a name, along with games, not being able to share a name with other games at the top of the hierarchy.

Inside of the GameService class we have the addGame method, which uses the iterator pattern to search the list of games and check the new game name against existing games in order to avoid duplication. The same pattern is applied in the addTeam method in the Game class and the addPlayer method in the Team class.

We can see the OOP principle of polymorphism applied in the GameService’s methods getGame, where the method used will be decided based on the signature of the input parameter, meaning the driver is able to call getGame with whatever info it has, be it an id or name, and return the Game by id.

**"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](#_z337ya)

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** | Apple has discontinued development of MacOS server variants as of April 21, 2022. Some server applications (file server, time machine) are bundled into the normal MacOS used on apple computers.  The fact that Apple no longer supports a server operating system means that while it is possible to run a web server on Apple hardware, it is not supported and may even violate Apple’s licensing for personal computer use. | The Linux kernel is open source, and is implemented in many fully complete operating systems that are also open source and free to use/license such as CentOS, Debian, Amazon Linux. Often these open source companies offer paid support for a minimal cost relative to many licensed products, for example RedHat Enterprise Linux.  The most popular web servers, the two most popular being NGINX and Apache web server, which are both available on Linux and supported by many open source distributions of Linux.  In addition all of the popular cloud providers provide base Linux images from many open source distributions. | Windows is a very capable operating system for running web applications, specifically if the code is dotNET, as Windows has first class support for all of their own products and integrates them together very well. It is simple to use Windows Server tools to install IIS (windows web server) and to target and build dotNET code and link it into the IIS setup.  The downside to Windows as a server operating system tends to be expensive licensing costs, with obtuse pricing models. There are also licensing costs associated with many other pieces of infrastructure that would run on, including the IIS server itself, and Windows’ SQL Server database. | While a mobile device can technically be used to host a webserver in the case of Android as it is a Linux derivative, it is not designed for this in terms of its hardware or software.  Resources are limited and processors are generally low power in order to be energy efficient on battery powered devices.  Common web server software is normally run on linux and windows, meaning support for ARM devices like mobile phones is almost non-existent, so in the case you *can* run it on a mobile device, company support likely doesn’t exist or is severely limited. |
| **Client Side** | MacOS users are likely to use the built-in Safari browser, development will need to consider testing and cost related to this. Safari is generally considered one of the major browsers due to ubiquity of MacOS devices, and the actual browser code is portable and not specific to Safari, so it is mainly testing for bugs related to using that specific browser. Developers who normally work with Apple devices are likely | Linux users use a large number of browsers, generally open source browsers like Mozilla Firefox, and Chromium/Google Chrome, with a number of other smaller browsers used by certain power users. Support for these smaller browsers may not be feasible as they are numerous and their number of users may be a small fraction of the total user base. | Windows users have historically had a split between the native browser (currently Edge), and other browsers like Firefox and Google Chrome. Since Windows makes up a massive number of PC users, and these other browsers are also present on systems other users might use like some MacOS users and many Linux users, testing can cover a lot of user base across multiple platforms by focusing on these 3. | Mobile device development generally requires specific expertise and the client libraries are specific to their device operating systems (Android and AndroidSDK, iOS and Swift/Xcode). This will require developers with that specific expertise, generally developers only specialize in one of the major operating systems, iOS or Android, so we would need expertise in each one which can get expensive. |
| **Development Tools** | MacOS apps are developed using the Apple software development kit(s) in the XCode Editor/IDE.  Swift is the language generally used for developing macOS apps, but the sdk also supports Objective-C. Developer experience is likely to be limited to devs who already work on this platform as most of the tools are non-portable, or if they are portable are not the first-class choice in other operating systems. | Linux supports development in a wide variety of languages as it is often simple to install a compiler for a language and build it using any text editor. There are many commercial IDE’s available for Linux as well such as the VSCode text editor with plugins, and JetBrains IDE’s for specific languages like CLion, IntelliJ, Goland, etc. There are a large number of developers with expertise in Linux. | Modern windows applications are usually developed using the .NET (dotNET) framework using the windows IDE Visual Studio. This IDE has tooling that integrates directly with Windows and the component pieces such as the .NET libraries, Windows API’s, and IIS web server. | Language support for mobile devices is tied specifically to the development tools for that OS, being the Swift language or Objective-C and the XCode IDE on iOS and Android Studio for Android. Each of these will also require developer expertise in both of those platforms and their libraries. |

## 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**: Linux is the clear winner in terms of operating platform, as it offers a large amount of flexibility (almost all software that runs on windows has a linux version, while linux has significantly more open source options available as well).

In addition to being flexible with many options for software that runs on the operating system, Linux itself has many freely licensed distributions that can be run at only the cost of having staff to manage them, or if needed there are companies that support their own distributions and offer support contracts like Red Hat. Linux images are available on many cloud providers which also opens up more options for hosting the servers in a way that doesn’t require staff to manage hardware and data center materials.

1. **Operating Systems Architectures**: As The Gaming Room requires that clients be able to use both PC’s to access the game on the browser as well as mobile devices that can access it with native apps, the proposed architecture is a client server architecture where the server or ‘backend’ will run on the linux server and the client side or frontend is one of a browser based app on PC, iOS app on Apple devices, and an Android App on Android devices.

This architecture allows separate development of all of the components, while allowing the backend to serve any clients that can use the REST API spec and HTTP protocol. The frontend then, can be native applications on mobile devices, or JavaScript running in the browser on PC’s.

The backend service can be written in many language options such as Java, as Linux is a common build target for most languages, and the web server code can be combined with a production web server or reverse proxy like Apache’s httpd or NGINX, both of which are available on linux and are high performance web servers that are capable of proxying requests to multiple instances of an application in order to get better resource utilization on a single server, or to send requests to multiple instances of a backend application in order to load balance requests to multiple smaller servers.

1. **Storage Management**: Storage management will likely depend on the specific choice of running on premises servers in a datacenter in which case hardware can be chosen for any number of requirements, or running on a cloud provider which generally provide generic options like magnetic spinning drive, and a few different classes of solid-state or NVME drives.

As our storage requirements on the server side are minimal, requiring only that we store our codebase and a small library of images at this point, we could implement a local copy of the library on the server(s) running the application which could use a solid state drive in order to have faster performance when loading images into memory, allowing the application to not need to store copies in memory when not necessary. At a later point if the library were to grow or the server infrastructure were to grow, a centralized storage solution like a SAN or storage area network could host the files which can be connected across the network to the servers. If we were to go the cloud provider root, images for example can be stored in a generic object store like Amazon Web Services’ S3 (Simple Storage Service), which functionally works like a SAN in that the data can be accessed over the network, but is even simpler as it is not a block store like a hard drive, meaning programs can treat the images as a file existing at a url path like mys3bucket.amazonaws.com/myimage. S3 handles versioning new files, backup, encryption, and even has stricter security for web based authentication using Amazon’s internal access mechanisms.

1. **Memory Management**: For memory management by the backend application, the operating platform does have some features that assist with using less memory over time, like swap files, which allow an app to write memory locations that are used less to disk in order to free up RAM. In this case we are likely not loading the entire image library into memory as mentioned previously in the storage section, so we will likely not use these features until the application has expanded.

On the client side memory management will be more important, specifically with mobile devices as they have a limited hardware footprint relative to a desktop PC, especially lower end mobile devices may have very limited memory. On the client side we can can limit memory use by only loading images as they are needed, but we will still likely need to store a copy of the library on the client and swap out current images in RAM until we reach the size or complexity necessary to use something like a Content Delivery Network to pull the required images from localized servers without introducing latency by pulling the library directly from the backend every time it is needed or taking up needless drive space by storing the library locally on every mobile device.

1. **Distributed Systems and Networks**: The basic setup can run on one server, hosting the backend, web server, database, etc. However this design is not the best for redundancy as a single failure will bring down the whole application. As mentioned before, running a webserver that can load balance multiple backend instances of an application means that if a single server were to fail, there are others that can serve the requests. We can implement this through the entire chain, having a load balancer like HAProxy in a high-availabiity pair setup, send requests to a web server like httpd that is setup in a similar pair, to application servers running the actual code that are setup similarly, and the same for the database that can run on it’s own set of servers with it’s own load balancer, etc. This setup is extremely complex to handle when managing our own hardware and networks, and would require expertise in system administration and network engineering. Cloud providers like AWS offer managed services like their application load balancer, containerized application deployments, and managed databases that can reduce the technical overhead for very reasonable costs.

The other benefit of choosing a cloud provider for hosting is the ease of spreading our servers across multiple regions in case of a large network failure like an undersea fiber cable being cut or a natural disaster that removes power to a data center. Cloud providers have facilities that easily allow shifting our applications to another region in order to stay running in even large outages.

1. **Security**: The main security feature for the frontend application will be users’s authentication, which is submitted through the frontend and processed by the backend, storing hashed passwords and user information in the database. The REST API will return a request if a user is authenticated and various Java security decorators can be used to limit access to specific portions of the api that a user client would need.

In addition to the user based access, we can have roles enabled so that real users are limited in scope to the least privilege they need to play the game and manage their own account, while limiting administrative access to our own employees and those of The Gaming Room.

The REST API itself can be secured with HTTPS in order to prevent snooping of the requests from user devices that may contain authentication information. In addition, the previously mentioned web servers like httpd and NGINX are hardened more than the built-in web servers of many programming languages, and as mentioned in the distributed systems portion, we can limit access to the application to only the webserver process or host, making that server a hardened single point of ingress for traffic from users.

The operating system itself also provides many security features, such as process isolation with namespaces and c-groups (containerization), in addition to process ownership by limited users so that in the case of a breach on a single server hosting all the pieces of the application, that exploited process can’t access say the database without further privilege escalation. Linux also has multiple options for network firewalls, and modern intrusion detection systems that can scan logs, networks, and files for anomalous activity.