

“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 | 03/22/2024 | Kain Semonis | Everything that isn’t template. |

**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)

This document will outline the design considerations and logic behind the development of a web-based game application in which players will guess what an image is meant to represent, based on an image gradually rendered over the course of 30 seconds. The images will be drawn from a pre-provided library of images. The most logical solution seems to be one that includes a singleton pattern in order to be sure that there is only one instance of the game running at a time in memory.

## Requirements

1. The application must support multiple teams of multiple people playing multiple rounds of the game.
2. Each team must be able to have multiple players assigned to it.
3. Game and team names must be unique.
4. No more than one instance of the game can exist in memory at a given time.

## [Design Constraints](#_1ksv4uv)

Scalability: Scalability is important considering the number of potential players. A web-based application typically has relatively low requirements on the client side, but the load on the servers are likely to add up quickly. Therefore, the application should, in theory, include considerations for load balancing in order to distribute traffic as required.

Security: Logically, anything web-based needs at least a decent measure of security, not just immediately for the host of the services being given, but also for the clients that would be affected were they to connect to a compromised host. Additionally, DoS and DDoS attacks are a common enough occurrence now, that defending against them in some way is important, even if just by having some sort of anti-bot verification.

Compatibility: There is more than one web-browser in use today, so trying to be compatible with as many as possible is an important step in increasing availability to potential users. Even aside from browsers, being sure that the application works properly and consistently across different screen resolutions and sizes is important.

## [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)

Entity serves as the base class for other entities in the application, and contains the attributes id and name, and methods for ID and name retrieval.  
Player represents individual players in the game, and inherits the id and name attributes from the Entity class.

Team represents teams made up of players. It also inherits the id and name attributes from entity, but also has Players.

Game represents individual games played by teams. It includes attributes inherited from Entity, as well, and similar to Team with Player, Game has the attribute Teams, made up of the teams that are playing that game.

GameService acts as a singleton service in order to manage games, teams and players. It has the attributes games, nextGameId, nextPlayerId, and nextTeamId to manage instances of the game, as well as identifiers for those players, teams, and games. It has methods for adding games, retrieving games by their ID or name, and getting the current count of games, as well as getting the next available player and team IDs.

Games have a one-to-many relationship with Teams, as multiple teams can be in a single game.

Teams have a one-to-many relationship with Players, as multiple players can be on a single team.

Players have a one-to-one relationship with Teams, as, presumably, each player can only be on one team at a time.

**"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** | If development is done on Mac, it is more likely to be compatible with hosting a server on Mac.  The biggest likely advantage to hosting a server from a Mac machine would probably be between the publicity you would get from Ripley’s believe it or not for hosting a server on a Mac, or the increased security you would get because there is a relatively smaller amount of experience in defeating Mac-exclusive security systems. And I guess Apple makes peripherals that would make whoever is administering the server be maybe a little bit more comfortable? | Linux would probably be the best OS to use for server hosting from a theoretical perspective. Most linux releases are open source, and Linux has enough customization options that it would probably be the easiest to get the most out of any given hardware with Linux compared to the other options. Being open source also means that it would be very cost-effective, especially for smaller companies.  The biggest downside to Linux would probably come from lack of familiarity. While Linux is a pretty popular OS, Windows is the closest to being ubiquitous enough that I would feel comfortable assuming anyone working in a field like this would have at least some experience with it. | Windows biggest benefits would be between its simplicity, and its familiarity. Without actually finding any real numbers, it seems a safe bet that the vast majority of people that would be administering a server would have at least some experience on Windows, slightly, a small, but significant, number less on Linux, and another step down for Mac. This leads to Windows being the most developed-for OS for most use-cases; though, for server-focused software, Linux may actually pass Windows in this regard. Windows biggest disadvantage would probably be its having a relatively low ‘ceiling’ for what a user can do. Its ‘floor’ is also higher, in that essentially, fewer options are not given to a user, to keep them from breaking anything, while on Linux, those options are typically presented. | The biggest advantage to hosting on a mobile device would probably be cost. By not spending as much on actual dedicated server hardware, a company looking to host a server would be able to save money by doing it through a mobile device. Additionally, most mobile devices are designed around power efficiency, which would technically be a benefit; as would the fact that being a mobile device would mean it is a portable server, which may be useful if you are frequently being forced into new offices for some reason.  Mobile devices having limited resources in terms of both processing power and storage compared to actual server hardware is a large downside, and it would be compounded by iOS and Android’s not being optimized for server-related tasks. |
| **Client Side** | It’s likely that a program of this nature would require a MacOS dev kit, so licenses for each developer that would be developing for MacOS would be a consideration, as well as probably at least a few MacOS machines to test builds of the game on. Preferably as many such test benches as possible, testing across different versions of macOS and hardware configurations.  MacOS development would probably have the greatest discrepancy in expertise, as it often has the most polarized opinions, with one side swearing by it and us correct people refusing to touch it. So it’s likely that any developers that specialize in developing for Mac would have more exclusive expertise in doing so, but have less in other OSs, and developers that aren’t specialized in developing for MacOS are likely to have little to no experience in doing so. | Because Linux is open source, it isn’t likely that there would be a cost associated directly with developing for Linux. However, just because Linux itself is free, doesn’t mean that the tools used to develop on it are. So the costs for developing for Linux are likely to be harder to predict, based on the specific software the developers use to develop for it. | Similar to Linux, there is likely to be costs associated with IDEs and other developer tools when developing for Windows, however, assuming the developers want to not pirate Windows, there is a licensing cost for each Windows machine, so it is likely that the cost would be higher than Linux’s. | Similar to Mac, developing for Android and iOS will likely require licensing SDKs for the developers to use. Additionally, Android and iOS both have a more centralised place for software releases in the form of the Google Play and Apple stores, both of which place a higher requirement in terms of security, reliability, and price per update on mobile versions of the game than would be expected on Mac, Linux, or Windows. |
| **Development Tools** | According to Google, the most relevant programming language for Mac is Swift. Like most of the most popular things on Mac, Swift was developed by Apple, and so is likely to be the most specialized for developing for Mac. Similarly, Xcode is Apple’s official IDE for Swift, so it’d probably be the IDE used for developing on MacOS.  I will try to focus on more ‘exclusive’ languages for the sake of this section, or else each OS’s section would probably be about pointing out the fact that Python is consistently in the top 3 most common programming languages across platforms. | According to Google, again, the most common programming language for Linux that meets this criteria would likely be Ruby, and C++, and Go. For IDEs, Visual Studio is a good option, and Eclipse is also an option. | For Windows, without repeating myself about previous languages, the most common would likely be C#, and Java.  Visual Studio is still a good choice for IDE on Windows, and Eclipse is also still usable. Windows also has the advantage of the UWP platform, however, meaning that, in theory, any device developed for Windows 10 will function across the different types of Windows 10. There is probably a better word than ‘type’ of Windows to describe the fact that a standard desktop computer, a tablet, and an Xbox do all use Windows, but it isn’t identical, and I didn’t want to use ‘version’ here. | iOS devices typically still favor Apple’s Swift language, and also are still typically developed with the Xcode IDE. Android devices are most commonly developed for with Java, using Google’s official Android Studio IDE. |

## 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 this specific use-case, the best server platform would likely be cloud-based.
2. **Operating Systems Architectures**: Although a cloud-based system may not have the absolute best scalability at the highest end, in terms of scale-to-cost, it has other benefits that should more than make up for this. Because of a cloud-based system’s distributed nature, downtime of any kind is less likely to affect actual server uptime meaningfully. In the most theoretical scenarios, it’s likely that at some point the scale of the server systems being effectively rented from whatever company is providing it will reach a point where this company can no longer provide a great enough server capacity or resources to fulfill the game’s needs any longer; however, this would be at such extreme level of need that it is even less likely to develop before the company would have the means to no longer need to rely on a cloud-based system, and should be able to transition to their own hardware.
3. **Storage Management**: I would recommend a sort of two-pronged solution to storage. Assuming the 1.2GB library is even close to the same size as previously stated, standard, consumer-grade storage systems like basic HDDs and SSDs will more than suffice for both the present, and a good deal into the future, while also being an inexpensive enough solution that security-vaulting will still be easy to do. On top of this, I would recommend another cloud-based solution. Having an extra set of storage even if purely for the sake of redundancy would not be a bad thing, especially when dealing with fairly small storage quantities. Again, assuming the same or similar filesize for the library, even consumer-grade cloud storage solutions would be viable, but if the people in charge wanted to put in more time and effort instead of money but want to feel clever, they could create two different gmail accounts, and simply email the library to each other; in terms of scalability, it would probably quickly become more viable to have a bot do the emailing. This is, of course, a good deal of effort to avoid spending a fairly small amount of money, however, it is, in fact, a way to avoid spending money.
4. **Memory Management**: Most cloud-based systems have very good memory optimization and utilization. It is possible that, again, in extreme cases, you may get better memory management from a more ‘tailor-made’, locally-hosted solution, however what is likely to be, at best, a marginal increase in performance for a good deal of added cost and complexity would probably not be worth it.
5. **Distributed Systems and Networks**: There are probably few exceptions to the almost-definition of a cloud-based server solution being already distributed effectively around the world. The client-server model would probably be the most effective, simply having the client interact with the server through whatever APIs and similar resources are implemented.
6. **Security**: Again, most cloud-based systems will have a good deal of security measures in place. To add on to them, though, fairly simple user-authentication with username+passwords, and probably multi-factor authentication would be very good to implement. On top of those, logging the IPs of each login would be a good measure, so every login can have that IP compared against the last one associated with that user, to trigger potential further security measures.