

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 | 11/09/2023 | Justin Holmes | Document Creation, adding |

**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)

This application will allow users to play four rounds each being one minute, during which an image is rendered over 30 seconds and during that time the players should make guesses for what the image is. Then after the time is up the each of the other teams can submit one more guess within a 15-second time limit.

## Requirements

1. *Must be able to have at least one(1) team playing. No specified limit.*
2. *Each team could have multiple players assigned to it.*
3. *Each game name must be unique*
4. *Each team name must be unique*
5. *Users must be able to check what names are in use.*
6. *Only one instance of the game can exist in memory at any point.*

## [Design Constraints](#_2et92p0)

We will need a server side application to run that will handle processing requests from each user, this will store and coordinate central data to align information going between client users.

The application will also need a master clock to keep things in time.

The application will need some database of images to pull from, and a method to render them over 30 seconds.

The application will need to allow guesses from the active team during the 30second timer.

During the 15 seconds the other teams can submit their guesses.

## [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)

This program will have a Driver with public function main() that will import a Singleton Tester to use testSingleton() to check if the game is already running somewhere. Additionally there will be a class Entity that will serve as a base Class for Game, Team, Player. It should these variables have a private long called id, and a private String called name, as well as these functions: private Entity() constructor, public Entity(id, name) constructor, getID(), getName(), toString(). The class Game which builds on Entity should have a private List of Teams, and public functions, Game(id, name), addTeam(name) which returns a Team class and toString() which returns a string. The Team class should have private variable: players a list of players, and public functions: Team(id, name), addPlayer(name) which returns a Player, toString() which returns a string. The Player class these public functions Player(id, name) and toString() which returns a string. The Game, Team and Player classes should be associated with a none to many link as well as the GameService class. The GameService class should have private variables: games which is a list of Game classes, a long nextGameId, nextPlayerId, and nextTeamId as well as service type GameService. Then it should also have a private function GameService() which is a constructor as well as these public functions, getInstance() which returns a GameService, addGame(name:string) which returns a Game, getGame(id:long) which returns a Game and one for getGame(name:string), a getGameCount() function that returns an int, a getNextPlayerId() and getNextTeamId() that both return longs. The GameService class should have a none to many associate with the Game class.

**"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** | -Pros, stable OS, Unix based, less vulnerable to security issues  -Cons, not available in RU formfactor for production server. | -Pros, open source, stable OS build, deployable on RU formfactor for production server, built in web app tools  -Cons, | -Pros, stable OS, available in RU formfactor, typically license locked features.  -Cons, more vulnerable to cyber attacks. | -Pros, small formfactor, but not rack mount, built in battery backup.  -Cons, not opensource, no native tools for web dev. |
| **Client Side** | Macs have a higher security needs, needing more dev time. Smaller hardware type pool, allowing a developer to enhance performance for particular machines easier. Needs specific licensing and products to develop on. MacOS devices cost more ($1000+) | Could be running on anything so larger hardware support needs, harder to utilize hardware acceleration. Open source allows for better rapid development and is then free/low cost. Most popular OS. Linux machines are cost effective ($100 for Pi) or potentially less on AWS machine. | Completely unique os, no hardware standards to utilize hardware acceleration. Needs certificates and specific things for installing programs on windows. Widely used so it’s a good platform to use. Popular so users will be familiar. Cost for a laptop or device is relatively low($500) | Typically needs a desktop platform to develop code on, and then those can run emulators or you need to purchase mobile devices to test the app on. Large community support and easy deployment routes(Google Play, iTunes). Higher cost per device($600-1200) |
| **Development Tools** | Supports most IDEs to my knowledge, as well as associated languages. Better for Apple ecosystem application development(building things for iPhones) | Great for open source development, runs most major IDEs as well as most major languages. Great for testing most server applications. | Supports most major IDEs and coding languages. Not great for application testing, I have had a hard time with web servers. | Not great for IDEs and extremely limited language support. No built in terminal so you need to download one. |

## 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 would recommend a Linux based OS for the server-side application. Because of its low impact on the system’s resources and large support base. Linux makes changing and setting up a server to meet the security, and performance needs of the application. I have personally leaned towards an Ubuntu 22 Server version, but other Linux distributions would be similar.
2. **Operating Systems Architectures**: I would recommend a monolithic server-side application, it should function within a single kernel. Letting users interact through a web page as a server-client application keeps the whole application light weight and doesn’t need large variations for different operating systems the users might use. Using this architecture would allow for future scalability of the system so if there were 1,000 games being played it could be deployed on multiple servers to meet the demand.
3. **Storage Management**: A simple onboard RAID 0 or 1 storage array would suffice, we are not storing mission critical data or storing large amounts so no need for anything with high bandwidth or redundancy. The data we are writing would be logs, and reading the program as well as the database of image files. This living as a separate drive on the primary server would allow for future expansion to some for of network drive or share.
4. **Memory Management**: We can let the OS handle memory allocations, we should still follow good practice to free-up unused memory when possible. As well as over delivering memory needs would prevent space issues as things potentially accumulate in memory while the program leaves. Simply if the design calls for 4GB or memory the system should get built for 32GB or more of memory.
5. **Distributed Systems and Networks**: This application being a server-client application it will have some major network requirements including a firewall, port forwarding and strict network rules to prevent Web attacks. Utilizing a local network layer 2 infrastructure would be fine because requests will go from the Firewall direct to the primary server it doesn’t need to traverse any other segments of the network. I would make this interaction on its own VLAN to ensure traffic only goes to that server, for additional security having a firewall that can identify any additional threats and block them. We should also utilize a static WAN IP and register with a DNS server so users can access the game easily.
6. **Security**: Internet security is at the forefront of this application, to utilizing secure firewalls/gateways to ensure only the appropriate web requests get through to the server. AS well as utilize HTTPS, SSL encryption, CA certificates to prevent unwanted access to the application. Any data we are accessing should be stored locally and utilizing passwords and checking authorization multiple times to ensure only intended users have access, and hash tables to prevent unwanted access via snooping memory blocks. Using encrypted data can ensure that if the physical hardware is accessed it cannot be read from without the key to the system.