

<Name-of-Software-Application>

# **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 | 08/04/2024 | Brad Peterson | Comparison of different operating platforms for client-server relationship. |
| 2.0 | 08/18/2024 | Brad Peterson | Offered recommendations as to system archecture. |

**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 client, The Gaming Room, already has a working game “Draw It or Lose It” on Android platforms and is looking to expand to more operating platforms. Creative Technology Solutions (CTS) will achieve this by creating a web-based version of their game. Each game consists of multiple teams, each with multiple players so a client-server relationship will work to handle these inputs coming from varied platforms and provide a single client to host the game on.

The game design will follow a singleton design pattern to ensure only one instance of the game exists at a time. The multiple teams and players will be handled by an iterator design pattern to ensure there are no conflicts when sending input to the game.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | CTS could develop the game server on a Mac machine through Mac OS X or MacOS. The Enterprise level costs of these servers if quite high. There are far fewer choices in this environment to find hosting through. | By far Linux is the most commonly used OS on web servers. It’s light-weight in terms of hardware requirements has the cheapest licenses. The high adoption rate of Linux and the open-source nature of it also means there is a lot of support in terms of Linux integrations to other products and resources to pull from to aid in development. Most importantly these servers are have the highest reliability of all the platforms so our game server won’t be offline due to server constraints. | Windows servers are also much more expensive than Linux servers. They’re main draw seems to be built in integration to other Microsoft products as well as it’s ease of use as it follows the design of other Microsoft UIs that most people are familiar with already. | These have extremely limited hardware capability and should not be used in this way as it is not sufficient to serve many users across different platforms. |
| **Client Side** | Clients will be on a web browser interacting with the server but on Mac the native web browser is Safari. Special attention should be paid mostly to things like font choice to ensure it looks the same across all web browsers as different OS can interpret the same font slightly differently. Similarly, CSS can be rendered differently on a different OS. | Clients will be on a web browser interacting with the server. However, Linux operating systems typically use fewer fonts than Windows/Macs. This means we should be sure to do quality checks on all operating systems. | Clients will be on a web browser interacting with the server but on Windows we should expect users to be using Chrome/Edge/Firefox as the main web browsers. Again, Windows may interpret the same fonts/CSS differently depending on how they are rendered so the website should be viewed on all OS during development. | We already have an Android application we just need to make sure it can access our server and display results accordingly. This will be done in Java.  We do need to build out iOS support. This means development in SWIFT. We also need to be sure our UI design translates as it may not be a carbon copy of the Android app due to differences in native handling of fonts and different capabilities of each language/hardware. |
| **Development Tools** | Languages:  Objective-C / Swift /JavaScript  IDE:  Xcode | Languages:  C++/Java/Python/JavaScript  IDE:  VIM/Emacs/Notepad++/Eclipse | Languages:  C#/Python/JavaScript  IDE:  VSCode | Android  Languages: Java  IDE: Android Studio  iOS  Languages: Swift  IDE: XCode |

## 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 our servers, we should be running a Linux based platform. This minimizes costs, allows the most flexibility as far as choosing a server provider, and can be more tailored to our needs than other options.

For the client, the mobile applications are fairly limited in optionality and will be made with SWIFT for iOS and Java for Android. The web platform could be made with web programming language but I’m going to propose we use React simply because that’s the framework our team already has experience working with.

1. **Operating Systems Architectures**:

Since we are choosing to use Linux based servers, I think using a service such as AWS to host our servers will be a good solution and provide us with many advantages over setting up our own servers. For this size application I think using a service provider like AWS benefits from their scale and will be cheaper than purchasing and maintaining our own hardware. This allows us the ability to scale, better reliability of servers, and traffic monitoring tools to make better decisions about managing the game.

The client is dependent on which platform the user is playing from. But in any case, we will use client-side rendering. One aspect of this is we need to control that rendering of the image in a way that is consistent across platforms, regardless of server-client communication times to provide a balanced game environment. Additionally, requests need to be asynchronous so users can submit requests without being impacted by the requests of other players.

1. **Storage Management**:

Our storage for actual images and code base will be quite small with only the 200 images making up 1.6GB of data storage requirements. However, I would expect as the game scales, we will need a lot more storage for user data to implement our role-based authentication.

We should initially examine the amount of traffic the Android game received to get an estimate of expected user base on launch across all platforms and then allow for enough storage to accommodate a growth rate of 3 times that traffic since it is now on 3 platforms instead of just Android. Additionally, we should have a plan in place to scale this storage as we may need in case we rapidly exceed our initial data storage solution. This is another reason to use a provider like AWS that has hardware systems in place to allow for this sort of demand instead of needing to acquire and implement our own systems.

1. **Memory Management**:

Running one instance of the game requires very little RAM to preload the chosen images and execute the game itself. However, if we have many instances of the game running, we could run into issues keeping up on the server side. We can utilize client-side rending to minimize the amount of RAM required for our servers to scale.

By utilizing this client-side rendering we are leveraging the local hardware capabilities of each user. On game launch, we should select the 4 images that will be used for the instance of the game and send those to the client to be cached and accessed quicker when needed. By frontloading this data transfer, we ensure we can load everything into RAM and have it ready to be rendered. Also, if there is a wait time to load, users are more accustomed to this happening before the game has started than between each round. This does impose a minimum hardware requirement to run the game on top of whatever the platforms themselves need to operate like a web browser or background applications on phones. That said, modern equipment will have more than enough RAM capacity to allow for this relatively lightweight game so I don’t think this barrier to entry will impact user acquisition.

1. **Distributed Systems and Networks**:

Our game is on a client-server basis, and we will have little control over outages on the client side. But we can utilize cloud architecture for our servers by spreading them out across many different locations to minimize the impact and likelihood of a widespread server outage. Since we will use AWS servers, they already have the infrastructure to support this, and we wouldn’t need to invest in all the equipment required to leverage this architecture.

1. **Security**:

We will use a role-based authentication system to log users in. For the minimum viable product there will only be one role for users as Player. Following the least-privilege principle this role will be capable of reading from our server and providing carefully crafted inputs to the server as requests for information to directly interact with the game itself. They will be able to create an instance of the game, invite other players, and, of course, play the game.

In the future I think another role that we may look to add would be a Leader role that allows users to upload their own photos for use in the game so setting up a role-based system now will allow for that sort of flexibility later but is beyond the scope of this design document.

Login information will use a modern hashing algorithm such as the Argon2id algorithm to safely store passwords.

We will use a service like AWS for our server hosting, these services come along with built in firewalls to prevent unwanted access if we follow appropriate security practices within our development.