

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 11/17/2024 | Adam Morales | Changes in the Executive Summary, Requirements, Design Constraints, Domain Model, Evaluation, and Recommendations section in this document. |
| 1.1 | 12/01/2024 | Adam Morales | Edited the Evaluation section of the document. (Server Side, Client Side and Development Tools) |
| 1.2 | 12/15/2024 | Adam Morales | Edited the Recommendations section of the document and edited responses to include more information. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has designed a mobile game called Draw It or Lose It which is based off of the 80s game show called Win, Lose, or Draw which has players guess what is being drawn. In Draw It or Lose It, the program generates stock images as clues for the players to guess and there are multiple teams that can play at once. Each game lasts four rounds at one minute each and it takes the program 30 seconds to fully generate an image. If the team doesn’t guess the puzzle before the time, then the remaining teams have an opportunity to guess.

## Requirements

The requirements that the client has requested are:  
Games must have the ability to have one or more teams involved.

Each team will have multiple players.

Game and team names must be unique.

Only one instance of the game can exist in memory at a time.

## [Design Constraints](#_2et92p0)

Some design constraints are that there can only exist one instance in memory at a time, meaning we must go out of our way to create unique identifiers for each game, team or player. Team names and game names, must be unique as well meaning we will have to check if a name is in use before allowing an identical name to be created. We also have to translate all the features of the mobile app into a web based environment which means that we have to take into account different hardware that each computer can handle.

## [Domain Model](#_8h2ehzxfam4o)

First, the Program Driver includes the main function which will be where we can run tests on the program when it’s complete, it uses the SingletonTester Object to help run these tests.

The GameService object is the largest object here and it is the main Game object that can only exist once. It holds key functions that create the game and get a variety of data from other objects. It can get the game instance from either an id or a name, and can get the next player’s or team’s id which the other functions use often for creating new teams or players.

The Game, Team and Player object all inherit from the Entity object which includes an id and name as well as functions to get and return both meaning we don’t have to recreate those variables repeatedly. This is a key feature in OOP and leads to efficient code that can be reused and modified easily.

**"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** | Good for server side development since it’s OS is based of UNIX, but recently some software specifically for running servers has been discontinued. | Incredible for server side development, is frequently used for running servers and it has an exceptional terminal with various commands to get the job done. | Windows can run servers and has great security features with its Windows Defender and good firewall. | Not recommended for servers since it must run at all times and uses a lot of energy. Phone would have to be plugged in all the time and be cooled adequately. There also aren’t many ways to run a server on a mobile device. |
| **Client Side** | Great for web development, has great tools such as VS Code which most web developers already use, and is very easy to pick up and learn for new developers. Many developers are comfortable using Mac as well. | Pretty good for developing client side, and it is a free OS which is worth noting, so the cost will be low, but the problem is not many people use Linux. | Windows is by far the best option for developing on the client side since a majority of people already develop on Windows so it will be very easy to develop since there are a lot of tools that can speed up the process. | Developing on a mobile device seems very inefficient. There is not much well known software for developing on a mobile device and if there is, it will be hard to find developers who prefer to work on mobile than on Windows, Mac, or Linux. |
| **Development Tools** | HTML, CSS and JavaScript will be the main development tools across all devices. For Mac Visual Studio Code is the best option. | Visual Studio Code or Vim can be used to write JavaScript, HTML and CSS but VS Code developers are more abundant than Vim developers. | Visual Studio Code is the best option and most web developers are familiar with it. Eclipse and Atom are also good options. | Code can be written in any text editor you want to, meaning any text editor on the App Store or Google Play Store is sufficient, but most devs, if not all, will prefer using Mac, Windows, or Linux. It is also inefficient to program on mobile because of the small screen and lack of keyboard. |

## 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**: My recommendation would be Windows since most developers are accustomed to using Windows and because Windows, by far, has the most amount of developing software made for it.
2. **Operating Systems Architectures**: The Windows architecture starts at the Kernel where it is responsible for managing all hardware and managing various scheduling processes as well as multitasking and resource allocation. There are also different layers of privileges so that a regular user cannot break the system but can still run everything an average user would ever need. There is a clear separation between the admin’s privileges and the average user’s privileges.
3. **Storage Management**: I would recommend storing the data in a cloud based server or an internal server. The internal server would be more secure, but with both, you can pull and push data to it and it will be running 24/7. Google’s Firebase or MongoDB are good options for a cloud-based server and there are a ton of documentation for both in order for developers to get started quickly. Google Firebase also has a good UI that makes it very easy to manage your server.
4. **Memory Management**: Windows will allocate a section of memory off in the RAM to run the game and only load processes when they are actually needed. If the memory is full which should never happen for this game, Windows will remove old memory in place of new memory. This game should run very easily on most computers that exist today.
5. **Distributed Systems and Networks**: If Draw It or Lose It will be cross platform, that means that all the various operating systems that it runs on need to pull data and push data from the same database so that they all work with different users on different hardware. Since all hardware will be interacting with one server, there are very minimal things that the developers need to do in order to make it cross platform since all of the data that will be used for the game will be stored on a server and all hardware shares the same data. All data will be pulled from the same server so there shouldn’t be much problems with cross platform functionality. Whether it is a Playstation playing the game, or a mobile phone playing the game, all the API requests to the server will fetch the same data, the only issue is figuring out how to display that data to the player with a good UI which is all client side anyways.
6. **Security**: Windows has built in security features that come with Windows Defender and it’s excellent firewall. However, it is important to note that if using a cloud based server, or a server running on other operating systems, never upload any important keys that allow access to said server and always hide sensitive data, most developers already know this and it won’t be an issue. The biggest issue is if the cloud server goes down, or if the internal server turns off unexpectedly.