

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 | 01/28/2024 | Dan Kleiner | Initial Documentation |

**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 Gaming Room is developing a puzzle game called Draw It or Lose it. The game will have players trying to guess what is being rendered. The staff at The Gaming Room is having difficulty setting up the environment. We should look at the current design.

## Requirements

* The game requires one or more teams to be able to play at the same time.
* Each team must have multiple players on it.
* Each Game and team must have a unique name and allow users to check if the name is in use.

## [Design Constraints](#_2et92p0)

The game is going to be distributed over the web so the team will have to have a server to host the game and that server must remain secure. The game must also have a way to store user information so that usernames and team names can be tracked to make sure that each name is unique. The application must let all users connect to one game that only has one instance loaded at a time.

## [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)

In the new UML Diagram, we see that we are going to implement a new class called Entity. We will be using this to implement the Factory Design pattern. Entity will be the parent class for Game, Team, and Player Classes. This will simplify the code in each of those classes. Once we create the class we can refactor the application to implement this 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** | Mac has stopped support for their server products in April 2022. | Linux servers are open source making it very customizable, to fit our needs. Linux servers also have a reputation for being very stable and dependable.  Linux is also the most cost-effective of the choices with no licensing fees.  Linux also has robust security.  Linux has limited amount software comparatively speaking. Linux may have a steep learning curve compared to other options. Gaming support is also limited compared to Windows. | Windows Servers are user friendly with a graphical user interface built in. They have extensive support from Microsoft. They also have support for monitoring and maintenance software.  Window does have a higher cost due to licensing fees that start at $1,069. They also tend to have more threats from cyber-attacks and malware. | Using a mobile device is not practical for hosting a server as they are not built for the amount of traffic that needs to go in and out. |
| **Client Side** | Mac Software Development Kit supports a variety of languages. There are fewer users for MacOS than other platforms. The Software Development kit in MacOS is in SWIFT or Objective-C which are Mac specific languages. | Linux has versatility when it comes to development as it is very customizable, and we can use many commonly used languages. Linux also allows for more security.  The drawback for Linux is that there are not many users for it in the public. | Windows has a super majority of the market when it comes to the market share of computers. Windows development is typically done in C# or the .NET framework.  As with the server Windows Security is not as robust as some options. | Android development is done with Java. While IOS is done with SWIFT so multiple versions would be needed to be created to get the full market. |
| **Development Tools** | Mac uses Objective- C, SWIFT and C++.  Mac does still have support for other languages and frameworks.  Xcode is the primary IDE for Mac’s but it also supports tools like Qt and wxWidgets.  Xcode cost $99 per year | Linux supports a vast array of languages from C, C++, Python and Java.  Linux also supports many different frameworks and APIs.  Linux development environments include GCC, Python and Qt and GNOME.  Many of the Linux IDEs are free to use | Windows primary languages are C++, C#, Java and Python.  Microsoft Visual Studio Code is a popular IDE because of its many plugins that can help aid development.  Visual Studio is priced at $45 per month or $539 for the year. | Andriod SDK is based on Java and is developed by Google. This can be developed in Android Studio which is free.  iOS is Objective-C and SWIFT based. It is developed using XCode and costs $99. |

## 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**: Draw it or Lose it will perform best if we use a Linux server to run the back end of the game. The front end of the game can be made in any language as Linux has many languages that are compatible, meaning that the clients can write in just about any language to connect with an API.

Linux also offers the most cost-effective because many distros do not have licensing costs.

1. **Operating Systems Architectures**: For the Architecture I would like to recommend a Client-Server Architecture. Using this Architecture will allow us to store user data on a server. Keeping the data on the server will allow for matchmaking services to run or for players to have friends lists. Using this will also allow for the game to scale, as images can be added, deleted or altered to keep the player base engaged. Running these services on the backend will allow the game to scale over time.

On the client side is where the users will interact with the game. This is where the rendering of the images will render, and the player will interact with the game. These processes will occur on the user’s device and send requests to the server.

1. **Storage Management**: The recommendation here if for Game Room to maintain a cloud server to manage the player database and well as a database images. Some sort of hardware may be needed, or cloud storage will be needed.

We can then only have the client download the images it will be using for each round from the server to save storage space on the users’ devices.

1. **Memory Management**: The server will use the memory to load the images that will be sent to the client side. It will also load in users as they are needed during matchmaking and send them to the client side. This will work similarly on the client side only one or two of the images will load into the RAM as to not use too many of the resources.
2. **Distributed Systems and Networks**: Since Gaming Room would like the game to be able to perform over the internet, I would suggest that we implement a RESTful API. A REST API will aid in the communication between the clients and servers even if the clients are using multiple operating systems.
3. **Security**: The security of the application should have multiple layers of user authorization. Most of the users should just have a player profile which will only give them permission to play the game but only change a limited number of settings that will only affect their user experience. Then there will be an admin level of permissions that will give the user access to change other settings like adding images to the server or changing firewall settings. Using REST APIs will also add to the security because it uses HTTP and support TLS encryption to secure data during transmission.