

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 05/22/24 | Matt Carrillo | Initial game design document |
| 1.1 | 06/06/2024 | Matt Carrillo | Information added to Evaluation section |
| 1.2 | 06/17/2024 | Matt Carrillo | Information added to Recommendation section |

**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 currently has an Android app called Draw It or Lose It available for download. They would like to develop a web-based game that serves multiple platforms designed around that current game idea. The game should contain multiple games which can host multiple teams and each team has multiple players. In order for the game to function properly, each game and team name needs to be unique. Draw It or Lose It contains a large library of stock drawings that it compiles for use. The staff at The Gaming Room are unclear on how to set up the software environment.

## Requirements

The game must be compatible on multiple systems/platforms. Each team will have multiple players on it, and multiple teams can play at one time. Only one instance of the game can exist at one time. Game and team names must be unique.

## [Design Constraints](#_2et92p0)

Android, IOS, and the web-based environment all have accessible software development kits.

The API should be tailored to work with these three different platforms.

The API also needs to be able to allow one or more teams from any of the platforms.

Game names must all be unique, as well as team names must all be unique.

Have the ability to alert the team captain that a team name already exists, and allow them to create another one. Use unique IDs for each instance of a game, team, or player in order to limit instances of the game to one.

## [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)

The Program Driver Class is where the main method is located. Program Driver uses directed association with Singleton Tester in order to test if there is already an instance of the Game Service. Entity class is the parent class to the Game, Team, and Player classes. Game, Team, and Player all inherit Entity’s required attributes, as outlined here:

* A Player can’t have a Team, but a Team can have a Player.
* A Team can’t have a Game, but a Game can have a Team.
* A Game cannot have a Game Service, but a Game Service can have a game.

Furthermore, Game Service can only have one instance of each game running at any given time and each Game can only have one unique Team. Each Team can also only have one individually unique Player at one 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](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac devices can be used as a hosting platform, given properly obtaining a license. It can be rather expensive though. | Linux is decently equipped for most server hosting requirements, and its licensing is free too. | Windows maintains fairly secure servers that are relatively easy to set up and use. However, like Mac, licensing is rather pricey. | Mobile devices are able to be used as hosting platforms, but are not designed to do it well at all compared to the other options here. |
| **Client Side** | Mac offers easy to use SDKs, which can lessen the necessary development time, but a Mac device is required. This adds to the cost of choosing Mac for developing and using overall. | Using Linux would take the longest in development time, as it would require the use of python, but financial cost would also be the lowest comparatively. | There are many aspects of windows that would require multiple different expertise. The .NET framework can improve the capabilities of the systems involved as well. | Development time and cost are the lowest on mobile, but the expertise is the highest as mobile applications are developed differently from computer applications. |
| **Development Tools** | A mac book that has iCode installed on it is required. All code would be need to be written by utilizing swift. | Python is usually pre-installed on Linux devices, so there would just be a need to utilize a python IDE, such as IntelliJ. | Visual studio is the standardized IDE for windows applications, and has the capability to use any coding language. However, most windows applications are written in some form of C language. | Mobile development has two standard options. Android apps can be made with android studio on Windows or Linux. iPhone apps are primarily made using swift in iCode on a Mac Device. |

## 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**:

Using Linux Ubuntu Server to host Draw it or Lose it on a Kubernetes cloud setup would be best.

**2. Operating Systems Architectures**:

The Linux kernel is regularly stable and the most secure. The Kubernetes clusters allow easy separation of systems and hardware needs.

**3. Storage Management**:

SSD storage is the better option over HDD. The SSD will allow for faster access to assets, as the game has to load pictures to users’ devices. This will improve the overall user experience. To optimize storage, a Kubernetes node for file storage and a NoSQL node for game data and user management is a great setup. In NoSQL, link the URL to the location of the picture.

**4. Memory Management**:

In order to lower the overall cost, there should be a watcher system supervising and managing the load the system is taking. This will allow only the necessary memory be taken up for the game to operate.

**5. Distributed Systems and Networks**:

Assuming that the system will utilize the cloud, if there is required miniatous on the servers your game will not have to stop. The node can be moved to another server, or if the system crashes another server will start up automatically and minimize down time. Going this route will allow for hosting everything needed to run the game, except for the client’s end. This will also allow all operating systems to have a client that can access the provided information. Separation of functions that Kubernetes provides make for easier management and organization the system.

**6. Security**:

Using a role-based security system is the recommended way to go. It will create separation from admin, game, team, player, and user. With this a user cannot access sensitive information that they shouldn’t be able to.