

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.gjdgxs)1

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.3znysh7)2

[**Executive Summary**](#_heading=h.2et92p0)3

[**Design Constraints**](#_heading=h.tyjcwt)3

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

[**Evaluation**](#_heading=h.2s8eyo1)3

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/15/2022 | Joseph Fultz | Initial design of software |

**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](#_heading=h.2et92p0)

The Gaming Room currently already has an Android app called “Draw It or Lose It” and they would like to develop it into a web-based game that can be supported on multiple platforms. The initial game itself consists of multiple games which have multiple teams that also have multiple players on said teams. For this to work properly each game and team name needs to be unique.

## [Design Constraints](#_heading=h.tyjcwt)

* Trying to tailor the software to multiple platforms raises concerns since each platform has different design kits for software.
* The software needs to be able to work with 3 different platforms.
* The software needs to allow 1 or more means from any of the three platforms.
* Game and team names need to be unique.
* The software needs to be able to alert a team captain if a team with the same name already exists and let them decide on a new name.
* Use unique ID numbers for each game, team, and player to make sure there is only 1 instance of each.

## [System Architecture View](#_heading=h.3dy6vkm)

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](#_heading=h.1t3h5sf)

ProgramDriver is the class that contains the main function for this entire software and uses Direct Association with SingletonTester to make sure that the game, team, or player is unique. Entity is the parent class to Game, Team and Player and those three each inherit Entity’s required variables. A Player cannot have a Team, but a Team can have a Player. A Team cannot have a Game, but a Game can have a Team. A Game cannot have a GameService but a GameService can have a game. Game Service must only have one instance of each game running at any time. Each Game can only have one unique Team at any time. Each Team can only have one an individual 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](#_heading=h.2s8eyo1)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as a mobile device, 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** | Macs can be used as a server but the licensing is expensive and you MUST have a Macbook to develop on. | Linux is very well equipped for a web-based situation. It is by far the most popular and licensing is free. | Windows has server capabilities and is very secure and easy to use. But like Mac the licensing is expensive. | Although mobile can be used for servers they are not well equipped to perform correctly due to the lack of power. Although, it can be utilized for development. |
| **Client Side** | Mac has a superb and user-friendly SDK, but the catch is that you need a MacBook to develop for Mac. This will increase the price and call for someone with quick development skills. | The most expensive part of this would be the development time; you also need a Python expert. | The most important requirement for Windows is experience. I would strongly suggest utilizing the.NET framework for its capability and security. | You should look for developers with experience creating apps for mobile devices. Different considerations than on the web must be made for user interaction and how information is displayed. |
| **Development Tools** | MacBook with iCode on it. | Python already comes pre-installed on most Linux machines. There is a multitude of IDE you could use for this. | Visual Studio Code is the standard for programming in Windows and can be done in multiple languages but C++ is usually the standard. | Regarding mobile, there are 3 possibilities. To create the app for Android, you'll require an expert in Android Studio. You need a Macbook owner who can use Swift in iCode to create software foriPhones. Or you may hire someone to use Unity to create the app. It is C++ and may subsequently be transformed into an iPhone or an Android app. To make it into an iPhone application, though, you will still need a Mac. |

## 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 should be hosted on Linux Ubuntu Server utilizing a Kubernetes cloud configuration, in my opinion.
2. **Operating Systems Architectures**: The Linux kernel is both secure and reliable. The Kubernetes clusters make it simple to separate hardware requirements from system requirements.
3. **Storage Management**: You can utilize either an HDD or an SSD for storage, although I suggest the latter. Because consumers won't have to load the photographs onto their devices as quickly, the SSD will improve the user experience. I advise setting up a Kubernetes node for file storage and a NoSQL node for game data and user management to optimize storage. The URL to the image's location is provided in the NoSQL link.
4. **Memory Management**: I advise installing a watcher for the system's load to help with costs. So that you only pay for what is required to provide the greatest experience, you can minimize the required memory while demand is low and add all that you need when consumption is at its height.
5. **Distributed Systems and Networks**: Due to the fact that your system will be in the cloud, even if there is server maintenance, your game won't have to be stopped. If the system crashes, another server will immediately start up, or you may simply migrate the node to another server. By choosing this option, you will be able to host all of the game's components except for the client side. This will make it possible to create clients for all operating systems that can access that data. Kubernetes' ability to separate functions makes it simpler to manage and organize your system.
6. **Security**: I advise employing a role-based security framework. The separation from the admin, game, team, player, and the user will best fulfill your needs. This allows you to prevent users from accessing information they shouldn't.