

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 | 02/18/23 | Tyler Smith | Initial prototype version of the software design |

**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 currently has an Android app of the game; however, they are looking to implement a web-based version that serves multiple platforms based on their current game. The client has no knowledge of how to set up such an environment. The game consists of multiple games, which in turn host multiple teams, and each subsequent team has multiple players. In order for the game to function correctly, each game and team name needs to be unique.

## Requirements

The client wishes to develop a web-based version of their current Android app. The client currently has no environment, nor the knowledge to setup such an environment, so will need a new one developed. This includes providing different options to allow them to choose from. The new app must be consistent with their current app, and also be usable from all 3 different platforms.

## [Design Constraints](#_2et92p0)

Android, iOS, and the web all have different SDKs. The API will need to be developed in such a way as to work on all 3 platforms. The API will also need to allow for 1 or more teams from any platform, however Game and Team names must be unique. There also needs to be a way to alert the team captain that a team name is already taken while allowing them the ability to choose a different one. Will need to use unique ID’s for each instance of a game, team, and player in order to limit the 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 ProgramDriver Class contains the main method. ProgramDriver uses Directed Association with SingletonTester to test if an instance of GameService is already in use. The Entity class is the parent class to the Game, Team, and Player classes. Game, Team, and Player all inherit Entity’s required attributes. GameService references Game, Game references Team, and Team references Player. 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 of each 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](#_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** | While possible to use Mac as a server, however licensing can be prohibitive, and you need to have Mac based tools in order to develop. | Linux is extremely popular for web-based hosting due to its open source nature. Licensing is free as well. | Windows servers are a viable method. They are secure and easy to setup. The licensing can be prohibitive however. | Mobile devices could be used as servers. However, they typically lack the power necessary for high end servers. |
| **Client Side** | Will require someone that has developed with Swift. In addition will require proprietary tools such as a Mac in order to develop. This is an added cost if you do not already have the tools. | Most likely would have a higher development time. You would also need to have someone that is proficient in python development. | Expertise would be the biggest requirement for Windows. You would need to use .NET framework for the security | Will require you to find developers that have experience developing apps. User interaction and the UI will have to be done differently than on the web. |
| **Development Tools** | A Mac Book with iCode installed. All code must be done using Swift. | Python comes preinstalled on most Linux distributions. I would recommend IntelliJ IDEA Ultimate for the IDE. | Visual Studio Code is Microsoft’s standard IDE. Any language could be used; however, the typical language for Windows is C++ or C#. | For Android you would need Android Studio. For iPhone you need a Mac Book with iCode. Another option is Unity which uses C++, you can convert to both an Android app and an iPhone app. However you would still need a Mac Book to convert to an iPhone app. |

## 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 is to use a Linux Ubuntu Server with a Kubernetes cloud setup.
2. **Operating Systems Architectures**: The Linux kernel is a secure and stable kernel that is open-source so typically when a vulnerability is noticed; the community will act on it rather quickly. The Kubernetes clusters allow for easy separation of system and hardware needs. It also comes with a wide variety of built in commands pertaining to the deployment and monitoring of applications.
3. **Storage Management**: I would recommend a SSD storage system. This allows for faster access to the library of assets than a HDD would, with the added benefit of being less likely to be corrupted. Combining this with a Kubernetes node for the file storage, and a NoSQL node for the game data and user management, would allow for quick access to the necessary files with the ability to scale the database if the need arose.
4. **Memory Management**: Linux has built in tools that allow for memory management overview. This would allow you to track the average load, which would give you an overview of how much memory is typically needed and adjust from there. This would allow for reduced costs if it came out that you needed less memory than initially thought to need.
5. **Distributed Systems and Networks**: With a cloud setup, you would have the ability to minimize downtime for the game. For instance, if there was scheduled maintenance for the game you could simply move the node to another server in order to keep the server up and running, or in the event of a system crash you could have it set to automatically shift to a backup server. This also allows you to host everything except the client side data, which would allow for multiple ways of accessing the information based on the operating system the client has.
6. **Security**: I believe that a role-based security system would be the best to implement for this application. With a role-based system, you would have the ability to separate the admin, game, team, player, and user roles. This would create a clear separation of what type of information a particular user should have access to.