

<Name-of-Software-Application>

# **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 | 03/24/24 | Peter Krentz | Initial Draft |
| 1.1 | 04/07/24 | Peter Krentz | First Revision |

**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 requests that CTS port their game “Draw it or Lose it” from Android systems to a web-based platform. The game needs to have multiple teams, with multiple players per team. Each game should be unique, and teams and players should also be unique.

## Requirements

* The game must be able to have one or more teams
* Each team will have multiple players
* Game and team names must be unique
* Users should be able to see whether a team name is in use when choosing their team name
* Only one instance of the game can exist in memory at any given time

## [Design Constraints](#_2et92p0)

The web-based application must deliver the same experience as the Android based application. We will use Java as our primary programming language. Since Java is used in the Android SDK, porting the application to a web-based service will take less time and require fewer rewrites of code. All current Android API calls will need to be reviewed and properly translated to web-based API calls if at all possible. We need to carefully select a tech stack and hardware that will meet the demands of the service without incurring additional costs to our client.

## [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 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.**

The GameService is a singleton that keeps a list of all active Game’s and upon creating new games it increments the GameID such that no two Game’s exist with the same name or ID. This same design pattern is used in the creation of Players and Teams so that each is unique from the other.

The ProgramDriver is the main application that the user interfaces with. It will call the addGame() method when creating new games, as well as the addTeam() method from the Game class, and the addPlayer() method from the Team class. When a new instance of Game is created, it is added to the games list that exists in the GameService class. Similarly, when new players and teams are created, they are added to the corresponding lists in the Team and Game classes.

The main loop would function as such:

* ProgramDriver initiates the addGame() method from GameService class. All instances of Game are stored in the games list in the GameService class.
* ProgramDriver initiates the addTeam() method from the Game class until all teams have been created. Teams are stored in the teams List in the Game class.
* ProgramDriver initiates the addPlayer() method from the Team class until all players have been added to teams. All instances of Player are stored in the players list in the Team class.

## [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** | There aren’t many providers for Mac based server hosting. The one that stands out however would be hostmyapple.com. It offers a decent amount of resources in a package that comes to about $25 a month. Their highest tier only comes in at $55 a month.  One primary advantage is that this specific service allows for development with Xamarin, which is a C# based development suite that can export to Android, iOS, and Windows. | Linux is the de facto standard for server hosting. The primary reason for this is that Linux is open-source. It is also quite easy for programmers to make custom additions to their Linux OS, allowing them to ensure the OS not only does exactly what they need, but ONLY does what they need.  If not hosting in-house, Linux database server hosting is available on nearly every database hosting platform. Some of the cheaper hosting services total up to around $15 a month. | While Windows databases/servers are not as widely used as their Linux counter-parts, in recent times they have risen in popularity, primarily due to Azure SQL, Microsoft’s personal product in competition with Amazon’s AWS.  While Azure has shown itself to be a competent competitor, it is only for businesses with cash to spare. With just the most basic features, monthly costs start at around $1k. | It is unfeasible to host a server on a mobile device. Mobile devices do not have the capability for the resources necessary to host a server for multiple users from multiple platforms. |
| **Client Side** | You will need an apple device that runs xcode in order to create a signed bundle for publishing the app for Mac users.  You can however avoid having to use xcode or swift for the actual programming of the client, and stick to Xamarin, which would be my recommendation as we can use this language to publish for all platforms.  Xamarin is an open-source platform, so the big cost is the apple device needed to publish the app, and the fee for the apple developer account, which is $99 per year. | Linux is open-source, and it can be installed as a secondary OS alongside Windows, meaning you may not even need new hardware for in-house devs. Linux is also rather “inexpensive” resource wise to run in a virtual machine for development (and tools like WSL exist so that you can do Linux development from your Windows machine).  As you may have guessed, Xamarin also works for Linux development. This means that the total additional cost for developing a Linux client is $0. | Xamarin supports both UWP (Universal Windows Platform) and WPF (Windows Presentation Foundation). This means we have a nice open source solution for developing the Windows client in addition to other clients.  Windows does have digital signing that you can apply for that will cost money, which can be anywhere from $400 a year to $800+ depending on the service used and whether or not online records are kept. You can distribute the executable without this signature, however, Windows will warn the user every time they run the program that it is not digitally signed.  Publishing the App to the Microsoft Store would incur a fee, ranging from $19 to $99, but is ultimately unnecessary as Windows executables can be downloaded from anywhere. | Xamarin supports both Android and iOS. iOS as mentioned in the MacOS section will require an apple device with xcode in order to create the signed bundle for publishing.  Publishing on Android to the Play Store will require a one-time fee of $25 for developer registration.  Play Store has some limitations to be aware of however. The app can only be 150MB or less. To increase the size, we will need to use Play Feature or Play Asset Delivery which allows the downloading of additional assets and such for apps that need to be bigger than 150MB. |
| **Development Tools** | Xamarin will be the primary tool of development, which uses the .NET platform and C#. This means we can use the Visual Studio IDE for development, although we will need to use xcode on an Apple device to publish the final application. | As stated, Xamarin and Visual Studio will be the primary development tools. We will of course need to test that the client application functions as intended on the Linux platform, which can be done via installing Linux as a secondary OS, or running it in a VM ala WSL. | Most of the development will take place on a windows machine, using the Xamarin platform in Visual Studio. Since we are developing natively on Windows, we can easily test our Windows client with no additional work arounds. | Xamarin supports development for both iOS and Android. Testing on Android is much easier as we can use an Android emulator such as Bluestacks or Nox and by putting our emulated device into Developer mode, we can run our own app without the need to publish it. |

## 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**: The big industry player for an operating platform would be Amazon Web Services, however, pricing can be tricky to nail down for exactly what you need. Instead, I’d recommend Digital Ocean. They offer all the services that will be needed to run the game, with highly transparent pricing. You only pay for exactly what you need, although, initial setup is not as simple as AWS.
2. **Operating Systems Architectures**: Linux will be the Operating System for the server-side setup. The user-side game client will run on the user’s local Operating System. While this means that an app must be made for each Operating System we intend to support users, this also means that we can guarantee a smooth and optimized experience for each user.

Offloading the front-end to local clients saves on resources necessary for the server and will save on costs as well. As for the backend, Digital Ocean provides support for both Docker and Kubernetes. Creating a container allows for easy deployment of additional servers should the need arise, allowing for automatic scaling. The server itself should give instructions to client’s, hold the list of game’s currently in progress and player’s online and connect player’s to games either existing, or creating new ones.

1. **Storage Management**: Digital Ocean provides both Volume Block Storage as well as Managed Database options. Storage on the server-side only needs to contain the database of player accounts. All image data can be packaged within the client-side application installer.
2. **Memory Management**: As the game will use the majority of resources on the client-side application, there is little to be done with memory management on the back-end server.

On the client-side, memory management will be handled by loading select images into memory, which can be called at the beginning of each game session. After each game session, images will be cleared from memory and new ones for the next game will be loaded into memory.

1. **Distributed Systems and Networks**: As the server is merely handing instructions to the client, all data can be exchanged asynchronously. Digital Ocean has many resources on implementing the restful API with their services.

As for outage protection, a cloud-based server platform helps to protect against outages by allowing for the deployment of new servers in the event that current server’s go down.

1. **Security**: The client-side application will only have access to certain functions for regular users. These should be; game creation, team name creation, team enrollment, and new user registration.

Power users, such as moderators or developers, will have access to more functions, such as removing users from a game, banning regular users, game deletion, and other moderation tools as necessary.

No client user role will have the ability to modify the server directly. All data sent to and from the client will be encrypted using the ECC encryption algorithm. This encryption method works well for TLS protocols, and should work fine for any client-server communication.