

# **Draw It or Lose It**

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/29/22 | Michael Reynolds | Edited Summary & Design Constraints |
| 1.1 | 08/08/22 | ^ | Edited Evaluation |
| 1.2 | 08/15/22 | ^ | Edited Recommendations |

**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 purpose of this software design is to develop a game loosely similar to ‘Win, Lose or Draw’. The client is in need of a streamlined development process for a web-based version of the game. Employing an SDK for web-based development and API integration will be the solution for this design problem.

## [Design Constraints](#_2et92p0)

**-A game instance will have the ability to have one or more teams involved.**

Since one team may be involved, the game may allow for non-competitive games without an opponent.

**-Each team will have multiple players assigned to it.**

A team is required to have greater than 1 player to be valid, otherwise that team’s participation must be void.

**-Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.**

Verify duplicate team names don’t already exist per game, and duplicate game names do not exist.

**-Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.**

Create unique identifiers and using the iterator pattern, ensure there are no duplicate games upon new game creation.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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** | MacOS has been moving away form server hosting, but with its UNIX based operating system, MacOS still is a viable option for server hosting. MacOS doesn’t offer hosting without a GUI natively but it is still a powerful and free option for non-enterprise low-scaling hosting solutions. | Linux is at the forefront of server hosting. It provides a faceless hosting solution, as well as powerful tools on its native CLI for server hosting remotely. It is suitable for scaling and enterprise hosting. General Linux server hosting is open-source available software. | Windows provides Microsoft’s server solutions, these include a licensing fee. Although the windows server hosting integrates well with many Microsoft applications, it is also limited in this way to more of a centralized solutions library. This provides well documented and beginner-friendly solutions, but with a cost and ultimately a dependency on the company itself. | Mobile devices in general don’t have sufficient computing power for hosting many clients, and given iOS’ strict SDKs and other mobile device limitations, this is all around a unaffordable and inconvenient solution. |
| **Client Side** | MacOS uses their native Safari browser. With differing support for HTML/CSS styling in comparison to Linux and Windows native browsers (firefox, edge), MacOS also tends to offer specific solutions for web applications. Across desktop web browser, they will all tend to integrate data from the API the same, but for application development, it may be handled differently. MacOS uses Swift as its main native application programming language. This is well documented but is also very exclusive to MacOS, so solutions with Swift may be considered more costly at times, and development time constraints may differ. | Linux platforms generally use firefox, a web browser with differing support for stylesheets used on chrome/safari browsers. The time constraints should be around the same, where development is less popular but picking up speed for orientation towards firefox. Expertise is long established with Linux development and cost will be similar to other desktop operating systems. | Development for windows is a more straightforward process when it comes to the kinds of solutions Microsoft provides for their OS. Microsoft provides beginner-friendly application development software that provides minimal coding solutions for powerful applications supporting HTML interfacing. Outside of the development software, most developers for windows applications will use powerful/well-adopted solutions, in this case most likely using a chromium based solution with thorough documentation for a client side application development. | Development for mobile device applications are definitely more costly. The average mobile app developer makes more than other platform application developers. The time can differ but is generally a bit longer than web applications given the curves of build/testing and deployment. As far as expertise, mobile applications are the most recent and prominent field to pique developer interests and there are many wide-ranging skill level developers for varying prices. |
| **Development Tools** | Swift is the native language of MacOS, the IDE is xCode. No licensing costs. | Although Linux is written in C, it is open-source and has many open source libraries for varying languages and IDEs to use as it does not provide a native preference solution for either. No licensing costs for any open-source solutions. | C++ / C# are the native languages for windows application development. Microsoft provides its in-house Microsoft Visual Studio for IDE. No licensing costs. | iOS uses swift and xCode, and Java with IntelliJ IDE or Eclipse IDE is the native language for android but many use Kotlin. No licensing costs. |

## 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**: When considering some of the top operating platforms, there are many to consider. “For the most part, the IT industry largely focuses on the top five OSs, including Apple macOS, Microsoft Windows, Google’s Android OS, Linux Operating System, and Apple iOS**.**” (INFORMATION TECHNOLOGY, n.d.) These are all great picks, but Linux operating system is my recommendation as it’s best used here as a free and secure standard for hosting the server.
2. **Operating Systems Architectures**: With Linux OS being open source, there are many architectures that may be considered. Many kernel options are given:

* Monolithic Kernel
* Micro kernels
* Exo kernels
* Hybrid kernels

Libraries, utility programs, hardware, and shell are all minimal and efficient to allow for strong computation speeds for headless server hosting.

(Linux Architecture – Detailed Explanation, 2022)

1. **Storage Management**: Using storage correctly, lets us take the load off memory, but it’s also important to consider that storage is finite. With a generally much higher capacity than the components that handle memory, we should focus on a decently sized server storage with fast read/write, something like a 120GB SSD. This will allow for overhead storage space, then we can utilize a database like SQL to store indexes for images and player data.
2. **Memory Management**: APIs and CDNs, as well as optimized caching methods, will ensure that we, much like in the described relationship between on-board memory/storage, utilize runtime memory and access certain data when needed from external storages over the web. So at the start of the game, we can preload the first 5 images, all picked randomly. These can be stored in storage on the device, and when the time comes for its use in runtime, it can be read/written easily from local storage.
3. **Distributed Systems and Networks**: Using RESTful API to keep our games in pace of play, we can ensure all player interactions our methods with proper requests. This means in consideration of connectivity issues, the client and server would be widely asynchronous, where the server will ping the client on occasion after expected requests to ensure connection. Should the client fall off, the server can notify other players of the disconnect, but resources won’t be wasted on constantly checking the client’s connection. The client will also have this same functionality to the server, ensuring the game is ended on the client side should the server not respond with certain pings. Utilizing multiple servers on multiple networks can help mitigate server outages, and instancing games in the form of deployed “microprocesses” on the server will prevent a server wide outage should one game fail.
4. **Security**: User information must be hashed and stored on the server, user data will never be stored on the client side and will only be sent through secure encryption to the server. User session will be handled with cookies which will be stored in client local storage, and cookies will be erased on the server and client at the end of/disconnection of each game. To prevent malicious payloads, all API methods will accept strict base64 encoding, and parsed for malicious code.

CITATIONS

**INFORMATION TECHNOLOGY, GWU, n.d. 5 Most Popular Operating Systems**

[Article] Available at:

https://www.wgu.edu/blog/5-most-popular-operating-systems1910.html#close

[Accessed 15 August 2022].

**InterviewBit** **2022**.

Linux Architecture – Detailed Explanation.

Available at:

https://www.interviewbit.com/blog/linux-architecture [Accessed 15 August 2022].