

<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 | <1/28/2024> | Cameron Ireland | Outline of operating platform pros and cons, as well as outline of design constraints. |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room, needs a web-based game that can be made available to multiple platforms. This game is called *Draw It or Lose It*. The game is currently available on Android, but the company wishes to make it available on other platforms such as MacOS or Windows. The game will always consist of a single instance of the game running, with multiple teams being in multiple sessions, allowing for players and teams to keep track of their statistics. This will require a two-part solution, with a server-side application, as well as a client-side application.

## Requirements

The client, The Gaming Room, is in need of web-based software that enables the most users to be able to access their program, *Draw it or Lose It*. Towards this, we must develop a two-part solution. This will include server software to handle the back-end operations, such as maintaining the singleton instance of the game and the stat tracking of each team and player. There will also be client-side software, most likely presented through a web-browser. As such, we must develop software that is compatible with Chromium, to ensure the maximum number of potential users can access the game.

## [Design Constraints](#_2et92p0)

1. Multiple teams and player per game:
   1. The game must be able to have more than one team, with each team having multiple players. With this, we will need to use a client-server architecture to ensure that the server can handle multiple users at a time.
2. Unique IDs
   1. Due to the nature of the multiplayer game, there are multiple players. To ensure that information is not lost per player/team, each player and team must have unique identification, whether it be a string of characters, a unique number, or some combination of those.
3. Cross-Platform
   1. Because this is intended to be a web-based service, it can be assumed that multiple people across multiple platforms should be able to access the game. Towards this, we must choose a web-browser type that allows for this. Our choice will be Chromium, as it is one of the most widely used in the market for common browsers such as Chrome, Edge, and OperaGX.

## [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)

Based on the UML diagram below, it can be seen that the Game, Team, and Player classes inherit from the Entity class. These three classes are also associated, as per the indication lines between them. The Entity class will contain the parameters of the id and name, as well as the methods of Entity(), Entity(id, name), getID(), getName(), and toString(). The three classes will inherit these attributes and methods, but each adds relevant information, such as a list of Players for the Team class, or the list of teams for the Game class.

Additionally, there is a GameService class which is associated the Game class, and provides the list Games, the nextGame, nextPLayer, and teamID attirbutes. Additionally, this contains the methods to add and get Games, get the gameIds, GameCounts, NextPlayerID and NextPlayer. Finally, there is a ProgramDriver class which utilizes a Singleton Tester to ensure that only one instance of the GameService is running 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 supports multiple server technologies such as Node.js, Python, and others. It can be easily integrated into Apple’s cloud service, iCloud, and the Unix environment provides a stable and secure environment. However, it has limited hardware options for servers, and has a smaller market share for game servers than Windows. | Linux is very cheap, due to it’s opensource nature, but comes with much more difficult configuration and customization, making the expertise requirement significantly higher than Windows or Mac | More expensive than Linux, less than Mac. Most common platform for games, has the most “skilled developers” due to Windows’ large market share, and has a huge variety of third-party tools | Due to the proliferation of mobile devices as well as native support for a variety of web-based software, the barrier to entry is low, but due to relatively low resources that vary widely across brands and models, there is difficulty in ensuring that all devices can adequately access the software. |
| **Client Side** | Development is more expensive than Linux, due to higher hardware and software entry costs, as well as expertise related to Mac compatible languages such as Swift or Objective-C is needed | Like Server-side issues, while the cost is almost non-existent compared to Mac or Windows, but the expertise required to develop client-side applications in Linux is less common for games. | As Windows is the most used in gaming, most clients will be able to support the platform with ease, with little to no issue with proficiency | Due to the ever-changing nature of iOS and Android OS’s, support will be difficult, with many issues arising frequently. This will be costly, and require a significant amount of time to handle. |
| **Development Tools** | Common Mac development tools include Xcode for the IDE, with languages such as Swift being only available on MacOS. However, there is little third-party tool support compared to Windows, and Windows/Linux development skills may not be easily transferable to the Apple ecosystem. | Open-source tools are abundant, there is native support for multiple common languages such as C++ or Python. However, the tools have a steeper learning curve, and there is minimal commercial tool support. | Common tools include Visual Studio, Eclipse, PyCharm, and a variety of third-party, commercially supported products, making development in C++, Java, Python, and multiple others accessible. | Mobile apps can support a variety of languages, including Python, Java, C++, and HTML, as the development does not take place on the devices themselves usually, and instead are run from a Windows, MacOS, or Linux computer. |

## Recommendations

**Operating Platform**

For the Operating Platform of the server, I will recommend Linux. This is for a few reasons. Primarily, this is because of its lightweight nature, with the ability to only utilize aspects of the OS that are necessary. Additionally, Linux is open-source, and as such has a wealth of information available on configuration and documentation. Linux is commonly used for servers for these reasons, as there’s no need to have precious resources being consumed by consumer-grade GUIs and back-end processes.

**Operating Systems Architectures**

Linux follows a typical monolithic kernel architecture. The system services, drivers, file management, and process management all run in the same kernel space, enabling them to share address space and privilege. Linux also separates its address between kernel and user. Critical systems are executed in the kernel space, and the user processes, such as applications, run in the user space. This ensures that user processes can’t destabilize the system by accessing kernel resources directly. Linux also utilizes virtual memory management, dynamically allocating and deallocating memory pages as it needs them. This will prove vital in a shared server environment, enabling maximum stability when multiple game instances are run on the same machine.

**Storage Management**

For storage management, Linux utilizes ext4, the default file system. This has many benefits, including journaling, large file support, online resize, and compatibility with previous ext file systems. Journaling lets Linux keep track of a log of changes before they are committed to the main file system. This ensures that in the event of a crash, or other catastrophic failure, the data can be recovered. Ext4 can support file systems up to 16 terabytes, making it an excellent option for games such as *Draw It or Lose It* that are heavy on images and other large files. Online resize allows for files to be resized while they are in use, without interrupting operations. This makes managing a shared resource environment much easier. For compatibility, ext4 can migrate and operate with pre-existing Linux systems, such as ext2 and ext3, which suggests that when/if ext5 is available, the server will not need to migrate its entire filesystem to the new system.

**Memory Management**

Linux utilizes many management techniques for memory. Demand paging, virtual memory management, and memory mapping. Virtual memory, as mentioned earlier, gives each process its own address space, independent of the physical memory. This allows for data to be swapped between RAM and hard storage easily. Demand paging allows Linux to ease the resource load by loading data to memory only when necessary. This frees up RAM that is otherwise usable. Memory mapping allows for processes to map files and devices to a specific address that they can directly access. This enables data exchange between processes without explicit read/write operations. This will be useful for *Draw It or Lose It*, as the game requires large datasets to be quickly accessible.

**Distributed Systems and Network**

Linux also provides excellent support for distributed systems and networking. The game can utilize technologies such as Remote Procedure Calls, Message Passing Interface, and even web services to handle platform communication. Linux also has built in networking capabilities, such as TCP/IP stack, and supports protocols like SSH and HTTPS, which provide additional communication security.

**Security**

Linux supports a variety of native security solutions, such as built-in firewalls, SSH, and regular security updates, depending on the distribution chosen. Other features include user/group permissions, file systems permissions, and access control lists. This can ensure that only authorized users are able to access potentially sensitive information. SSH allows for secure connection to remote servers by encrypting communication, which can stop man-in-the-middle attacks and eavesdropping. Other possible features with Linux include Intrusion Detection and Prevention Systems (IDS/IPS), which monitor network traffic and detect malicious activity. This can help stop any ongoing attack as it occurs, potentially avoiding costly breaches.