

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

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 | 11/12/2022 | Nicholas Nevins | Filled in information for Project One |
| 1.1 | 11/20/2022 | Nicholas Nevins | Submitted Requirements |
| 1.2 | 12/10/2022 | Nicholas Nevins | Submitted 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)

Draw It or Lose It needs a web-based version of their app and have reached out to us to provide the environment for it to function. They will need ways to ensure team and player names, and game instances are unique and able to join together to be in games together. This can be done through iterator and singleton patterns, as well as utilizing super and subclasses.

## [Design Constraints](#_2et92p0)

* App must be able to validate that player names, team names, and game instances are unique
* App must be able to join players into teams, and teams into games so that unique players can play games with an against each other
* Unique identifiers must be assigned to games, player, and team in order to assure unique instances

## [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)

We have the ProgramDriver, which holds the main() method, and it utilizes SingletonTester, which only serves as a testing module to ensure singletons are utilized properly. Main() will, of course, run the entire program.

Entity is a super class that houses the framework for Game, Team, and Player, all of which are derived subclasses of Entity.

GameService is a class that utilizes Game, Team, and Player in order to create lobbies for the game to utilize.

Because main() only interacts with GameService, which in turn only interacts with Game, Team, and Player through their own methods, we see abstraction at play keeping the user from being able to directly influence things they should not.

The three subclasses—Game, Team, and Player—are examples of inheritance, utilizing the base structure of Entity in order to branch off and accomplish slightly different tasks without the need to duplicate code.

The use of private and public methods is an example of encapsulation, restricting access where needed to the individual classes.

Polymorphism is best illustrated by the use of iterators and lists. Because the program functions with any player, team, or game—depending on context—it is designed to be able to run regardless of what input specifically is given. A team is built from players, but it doesn’t have to be certain players.

**"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** | Mac is going to come with licensing costs, and that can be expensive. However, it is very stable and has a lot of Unix features built in which makes it a solid option. It also sports a polished GUI to interact with the server. | Linux is open source and free to use. It is powerful, stable, and can run for days with little issue. It too has strong Unix features, and is great for servers. However, it tends to be harder to interact with and could be more difficult to use for less technically inclined individuals. Linux is arguably the most secure OS due to its open source nature. | Windows will also come with high licensing costs. While historically less stable than other options, it has improved in more recent releases. It does feature a smoother GUI experience than Linux, and is easy and familiar to use for most users. Windows has strong security. | Mobile devices tend to be more purpose built, and that purpose is hardly meant for servers. They lack the storage and hardware capability to truly run a server, and in my opinion should not be considered for server side purposes. |
| **Client Side** | Mac is superb at running web apps, and is highly optimized for a good client experience. Mac has a ton of built-in features and hardware to aid in development. Macs are more expensive, and will increase development costs. If you have a windows server, Macs are a bit more difficult, but that can be remedied. They are great for developing for Mac or Linux based servers however. | Once again, Linux is free and that is great for clients and developers. Developing on Linux is great for Linux servers, as developing on other OS’s for Linux servers can be a bit more difficult. Linux is stable, secure, and it’s open source community is great at keeping it that way. | Windows is highly familiar and extremely easy to use on the client side. It’s great for web apps. If you’re writing for a Windows server especially, Windows is a great tool to develop for. It is still a good choice if you are running a different server, but the OS is a bit more limited than other options. | Mobile devices are great for client side web apps. They play very well, often have built in touch screens and other options to make a smooth, clean UI. It will run great, and app markets are filled with web-based apps. However, like with running a server, developing on a mobile device is less than ideal. It often lacks the features and power to really do good development that other OS’s can offer. |
| **Development Tools** | Mac is Unix based and has a built in terminal, which is great for development. It includes an abundance of tools to aid in development right out of the box. It also includes functionality to run Windows and Linux on it, which makes developing and testing on those platforms much easier. Xcode is the built-in IDE, and is great for developing Apple apps. | Linux comes prepackaged with most compilers and interpreters that you might need, and any you lack can be easily nabbed through the command line. It also includes a great suite of tools to aid developers, like a built-in package handler. Lots of IDEs to choose from. | Windows supports an impressive amount of languages, and is the native OS for Visual Studio, an industry standard IDE. It has tools such as PowerShell to improve the Command Prompt. VS Code helps cover languages that Visual Studio doesn’t. It has a built-in package manager, and you can code for Linux right on Windows 10. | I would not recommend development on mobile devices. They tend to be purpose built for users, not developers and lack tools to aid in development. They are useful to have around for testing, however. |

## 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**: Though Mac is a strong option, my final recommendation is that we look to a Linux server. It requires more expertise from developers to interact with, but I think it will give the most bang for the buck. It’s cheaper, and it’s powerful. It possess a lot of built in functionality, and it has the security, stability, and flexibility that should allow *Draw It or Lose It* to grow in any way you might need.
2. **Operating Systems Architectures**: Linux is built in four layers:   
     
    **hardware -> kernel -> shell -> applications**

It’s set up that interacting with the hardware via the kernel is a fairly direct an efficient process for a user. The shell helps to keep interactions ambiguous enough that a user can’t do more damage than they should, and it utilizes resources well in order to keep applications running at peak efficiency.

As for out design architecture, I like a microservices pattern. I find it an elegant, efficient, and practical approach to dividing work and creating secure, modular, scalable systems.

1. **Storage Management**: I recommend away from Cloud storage and more for physical storage. I think the costs are potentially less, and the control you will have over your storage is superior to the benefits offered by cloud storage.  
     
   I think that a smart approach to partitioning drives into specific sets of images could help in order to speed up load times.
2. **Memory Management**: For best results, we’ll want to load images directly from memory instead of storage. I think to best accomplish this, we might find a way to keep a rotation of a set number of images in memory at a given time. If multiple games reference the same image, it will cut down on how many images must be moved from storage to memory. When all games have finished with a single image, it can be released from memory and a new one can be rotated in from storage.
3. **Distributed Systems and Networks**: For this we will clearly want a client-server pattern. This keeps the client devices and the server effectively disconnected from each other, and instead utilizes requests to the server in order to populate needed assets for the game. There are effectively no dependencies, so long as the server is running. A REST architecture is likely our go to.

A lightweight directory-access protocol is likely a good and industry standard option for login management. It is a secure method for validating user credentials and allocating free and open resources while centralizing the data needed by the server to make those decisions.

1. **Security**: Much of our security concerns could conceivably have been addressed in our earlier recommendations. A client-server pattern limits how much a client can access a server, while REST can be used to authorize users only to access what they should be able to access. A form of user validation, such as the LDAP protocol mentioned previously will help with login security. It’s also vital that we utilize operating systems that are current and designed with security in mind in order to limit security threats.