

Draw it or Lose it

# **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 | 7/19/2022 | Trevor Mayo | Added entity class that holds id and name values for classes. Added ability to create new games, teams for games, and players for teams. Added test cases for a print out. |

**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 is looking to develop a web-based game that serves multiple platforms based on their current game. The game is played by one or more teams and contains a library of pictures. The goal of the game is to guess what the picture is as a random picture is pulled from the library and fully renders over 30 seconds. If the team doesn’t guess the puzzle before the time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit.

## [Design Constraints](#_2et92p0)

- The game must run on multiple platforms.

- The game will have the option to have one or more teams involved.

- Each team will have multiple players assigned to them.

- Game and team names must be unique to ensure only one instance of the name exists

- Only one instance of the game can exist at any time.

The game must be written in a language that can run on multiple platforms. We must define the application to require a minimum of one team when trying to create a game. Each team should also have a minimum of two players assigned to them. We need to ensure that we check the names of all the teams (iterate list by name) when creating a new team to make sure there are no duplicate names. By using the singleton pattern, we can make sure that only one instance of a game is open at any time.

## [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 UML class diagram dictates that for the one Gameservice instance, there can be many games. However, there can only be one Gameservice instance . For the Game instance, there are 0 to many instances of the Team class. This means that every Game instance can have many Team instances associated with it. The same goes with the Player class to the Team class. This allows many instances of Player to a Team class. Team, Game, and Player all inherit Entity and it’s variables/functions. For example, this means that you could call “getName” for any of the 3 classes (Team, Game, Player). Those 3 classes will extend Entity in Java. Gameservice will use the singleton instance so that only one instance exists of it. The driver uses the SingletonTester Class which contains a line of code that instantiates one instance of the GameService class.

**"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 has faster access to the root of the OS, which allows a speedier hosting environment for a web-based software application.  Mac OS server costs start at 30.00. It’s very simple to download Lion on Mac OS to initialize a server. | Linux is extremely good for server-side speed when hosting web-based software applications. Very similar to mac. Linux is open source and free, so server costs are null. | Windows has much more support regarding it’s general os, but isn’t very popular for web-based hosting and has limited options for software development. Windows is also very expensive regarding their hosting. The costs can range from 15 dollars a month to over 1000. | Mobile devices are much slower as a server-side based host. The mobile device wouldn’t be attached to anything with ethernet, so it would have to receive and send data wirelessly (slower). The only benefit would be the ability to bring the mobile device anywhere. The server would be free. |
| **Client Side** | We’re probably going to write the program in a Javascript library. The overall cost/time/expertise to support an HTML based application would be the same with every client because it’s written in Javascript and not a different language for each client. Every major browser that you see nowadays has Javascript compatibilities, allowing the application to run on any popular modern browser. | We’re probably going to write the program in a Javascript library. The overall cost/time/expertise to support an HTML based application would be the same with every client because it’s written in Javascript and not a different language for each client. Every major browser that you see nowadays has Javascript compatibilities, allowing the application to run on any popular modern browser. | We’re probably going to write the program in a Javascript library. The overall cost/time/expertise to support an HTML based application would be the same with every client because it’s written in Javascript and not a different language for each client. Every major browser that you see nowadays has Javascript compatibilities, allowing the application to run on any popular modern browser. | We’re probably going to write the program in a Javascript library. The overall cost/time/expertise to support an HTML based application would be the same with every client because it’s written in Javascript and not a different language for each client. Every major browser that you see nowadays has Javascript compatibilities, allowing the application to run on any popular modern browser. |
| **Development Tools** | To deploy this application on Mac OS, we are going to be running this application in a browser, so it’s not necessary to have anything directly compatible with Mac, only the browser that it runs. A good IDE for building this application is VS Code. I say we use a full stack of React, Python, Rest, and SQL Tables to create the application. | To deploy this application on Linux, we’re going to also be running the application on a browser. Again, the same full stack used in the Mac OS would work just as good, along with the same exact IDE. | To deploy this application on Windows, we’re going to be in the same boat as the other two OS’s. Same languages, same IDE. You can actually develop the application in the windows environment if you want, but you could honestly do it in any of the three desktop OS’s. | To deploy this application on Mobile, we would be accessing it with a browser like the previous 3 Operating Systems. The browser handles all of the rendering related to the javascript, along with any of the rest api calls. It would be difficult to develop on an android or ios operating system itself. |

## 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**: I’d recommend using Linux when creating the Gaming Room’s games. There is very fast server side rendering along with fast software development if you understand Linux OS.
2. **Operating Systems Architectures**: Linux is an open source Kernel based OS. There is a hierarchical file system that allows easy access to root and installations. The kernel is in charge of choosing major actions for the Linux OS. Linux also contains a command line shell that allows the user to communicate to the kernel. Multiple applications and services can be ran at a singular time as well.
3. **Storage Management**: I can use the standard partition method for storage management on Linux OS. The OS partitions a piece of the disk to allow for files systems to be put on them. The file systems are similar to a normal library with bookshelves and organizational practices. There needs to be a mount point (mkdir) on the system to map the file system to.
4. **Memory Management**: The Draw it or Lose it game requires there to be multiple users, games and teams running at the same time. To ensure that API calls are quick and efficient, the Linux server should have the state of each game, player, and team stored in memory for quick access. There would be a partition of memory dedicated to storing that information.
5. **Distributed Systems and Networks**: s  
   I’d say that distributing the software through React/Javascript would be a great way for the game to be accessible on all platforms. React is a front end library for javascript to help with quicker/simpler creation of UI components. The requirement would be to have Firefox, Chrome, or Safari as a browser to access the game. API calls to and from the Linux server would be the way for the React App to gather information about each game, team and player. To ensure that there is solid connectivity between devices, the Linux server should be big and fast enough to handle many users trying to connect with each other. There should also be a generator for outages in the area. If we wanted to go cloud based, you could host on a cloud based server, that is guaranteed a certain uptime and elasticity for connectivity/outages. If there were many players trying to log on to the game at the same time, a cloud based system would be able to expand accordingly.
6. **Security**:   
   Linux has built in security within its OS. There are built in measures within the kernel to detect and deter vulnerabilities/attacks. Linux is also open source, which helps with its security by being found and destroyed immediately after version launches. Using API keys and multi factor authentication when logging into the game will help protect user information from malicious attempts between various platforms. Having strong passwords to log into the server is also very important for admin purposes. Anything less than 12 digit passwords should be a no-go. Avoiding unnecessary apps on the server will also ensure that a limited number of apps are installed on the server, creating a more secure environment. Utilizing backups on Linux servers also make sure that in the case of an attack, critical data is at least saved elsewhere. To protect any user data, the biggest thing is creating multi factor authentication and SSH keys for admin logins to the server.