

Draw It or Lose It

# **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 | 05/21/2023 | Jeff Jansen | Developing the overarching design and concept for the game as well as addressing any issues. |

**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)

Client is requesting a game in which teams of players draw and compete to guess the correct drawing within the time limit. The client is requesting that this game be web based which will allow us to focus on the base requirements within the system. Problems which may arise with this is teams/player counts unique names for players and games, and limiting to one instance of each game to limit resources used within the game.

## Requirements

Client requires that there be at least one team or more with multiple players possible on each team as well. Game/Team names will have to be unique as well so this will involve checking already created instances of games for the requested team or game name. Creating unique instances of each and verifying that there is not already something in session which has the name attempting to be created.

## [Design Constraints](#_2et92p0)

Main constraints with this will be having this application usable in different browsers to allow for more availability to more players. This will include browsers such as Chrome, Safari, Edge and many others. Game and team name checking will be mandatory to prevent duplicates and issues which may arise from that.

## [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)

Each instance of Game, Team, and Player will have inherited qualities which are set within Entity. The GameService with contain zero or more Game, each Game will be composed of zero or more Team, while each Team will have zero or more Player. The ProgramDriver will contain the main aspect of the games and will help to limit with the use of SingletonTester to prevent multiple instances of the GameServices class running.

**"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 server-based deployment based on the macOS operating system, offers server-based deployment options for hosting websites. It includes features like Apache web server, PHP, and other tools for web application hosting. Costs associated with this will be the one-time purchase from the Mac App Store. | Linux server-based deployment distributions like Ubuntu Server and CentOS are widely used for web server hosting. They support popular web servers like Apache HTTP Server and NGINX, providing flexible and scalable deployment options. The cost of this will depend on the support and maintenance services requested since there is no direct cost associated with the operating system being open-sourced. | Windows server-based deployment provides robust server-based deployment options for hosting websites. It supports Internet Information Services (IIS) as the web server and offers various deployment tools like PowerShell and Web Deploy. Licensing costs for this will vary and depend on the number of instances of that server. | Cloud platform server-based deployment provide server-based deployment options with scalable infrastructure. They offer various services like virtual machines, containers, and serverless computing, making it easy to host web applications. Cloud platforms typically charge for the resources consumed, such as virtual machines or container instances, rather than licensing costs for the operating system itself. The pricing can vary based on the chosen instance types, storage, and network usage. |
| **Client Side** | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mac.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Linux.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Windows.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mobile Devices.> |
| **Development Tools** | Programming languages supported on the macOS platform are PHP, Python, Ruby, Node.js, Java and are typically built in Xcode, Visual Studio Code, Sublime Text. The development team working with macOS Server would require similar language proficiency as Linux-based development. Xcode is the primary IDE for macOS development, and tools like Visual Studio Code and Sublime Text can also be used. The impact on the development team would be the need for knowledge of macOS development practices, Xcode, and related frameworks. | Programming languages supported on the Linux platform are PHP, Python, Ruby, Node.js, Java and are typically built in Sublime Text, Visual Studio Code, Eclipse, NetBeans. A development team working with Linux-based servers would need proficiency in one or more programming languages like PHP, Python, Ruby, or Node.js. The choice of tools and IDEs may vary based on the selected language. The impact on the development team would be the need for expertise in Linux server administration, command-line tools, and familiarity with open-source development practices. | Programming Languages which are supported on the Microsoft platform include C#, ASP.NET, .NET Framework and are usually built in Microsoft Visual Studio, Visual Studio Code, IIS Manager. A development team working with Windows Server would typically require expertise in C# and ASP.NET development. Visual Studio and related tools provide a rich development environment, debugging capabilities, and integration with IIS. The impact on the development team would be the need for knowledge and experience with Microsoft technologies. | Cloud platforms generally support multiple programming languages, including those mentioned earlier. Cloud-specific tools and SDKs provided by the respective cloud providers (e.g., AWS SDK, Google Cloud SDK, Azure SDK), along with popular IDEs like Visual Studio Code and Eclipse. Development teams working with cloud platforms need to understand the specific services and tools provided by the chosen platform. Familiarity with the respective cloud provider's SDKs, APIs, and deployment tools is essential. The impact on the development team would be the need for expertise in cloud architecture, infrastructure management, and integration with platform-specific services. |

## 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 would recommend that Asp.net is used to develop this application based on the clients requirements. This will allow for a highly scalable cross platform development.
2. **Operating Systems Architectures**: With this layered architecture the client will be able to easily, efficiently, and dynamically expand their game to the users who will want to play their game. Keeping all of the code reusable where necessary as well as allowing for development and use from many different platforms.
3. **Storage Management**: Storage will be dealt with on the server side of things and will only need to hold the information for the current game load with any long term storage unnecessary for a game of this type. This will help reduce load times while the program is checking for game or team names which already exist.
4. **Memory Management**: Asp.net manages its memory dynamically based on time of last access and will purge any memory which has not been accessed over a certain time frame freeing it up to be used in other places. This will keep the servers running efficiently without memory being held by unused aspects of the software.
5. **Distributed Systems and Networks**: Users will be connecting to the server with their end points which will distribute the game to them based on their internet connection. Doing this there can be issues with that connection on the server side or client side though that will limit the delays and such in game when doing a peer to peer type system keeping the timers and such consistent for all players. Down time will include internet outages as well as server updates, clients will be connecting through a browser so they will not have to deal with updates on their end of the application extending down time. This can be limited further with server redundancy and staggering the updates across the servers.
6. **Security**: With only active storage being utilized and not having to save game, team, or player data long term this will limit security issues over the long term since things will not be stored they will not be accessible. Also Asp.net applications can use multiple authentication modes such as Windows, Forms, or Passport just by adding a simple entry in the code.