

Guess That Drawing

# **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 | 05/23/2024 | Rachel Stevenson | Completed first draft of executive summary, requirements, design constraints, and displayed/described the domain model. |
| 1.0 | 6/23/2024 | / | Completed recommendation based on research. |
|  |  |  |  |

**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)/

We are coding an application for CTS that will allow 1 to several teams made up of multiple people to play an instance of the game “Guess That Drawing”. No team can have the same name, and only one game will be run at a time.

## Requirements

Teams will have 15 seconds to guess the image. This is a web based game.

1 or more teams involved. Multiple players on each team. Game/team names must be unique (users can check team name to see if in use). Only one instance of game can exist at a time (singleton).

## [Design Constraints](#_2et92p0)

Having only one instance of the game running at one time will be handled by the singleton design pattern. With this in place it will allow us to classify things like the players/teams into easy to use categories and solving problems like duplicate names will be easy. Since this application is currently just web-based we will not have to worry about things like creating special UI for varying phone application environments.

## [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 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.**

The program driver that contains the main code walks us through a loop for game play and also uses a ‘test singleton’ class to test that the singleton design in GameService is operating correctly. Game, Team, and Player all inherit data from Entity (which is simply a class that helps us assign an id/name to any type of one of those three classes). A game can have 0 to many teams and each team has 0 to many players. GameService does not inherit anything from Entity, GameService can create an instance of the game and then can run 0 to many games underneath this instance.

## [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** | Pros: Includes terminal for system admin tasks,  Tools available for management of software like Homebrew,  Apache is a built in web server, Integrates into apply easily.  Cons:  Not for large scale server traffic, has a higher cost for startup server hardware. | Pros: Very accessible with a lot of programs/tools available, cost effective, easily change settings on server security, supports languages like python and ruby.  Cons: If moving existing data to expand game server can be difficult to switch to linux | Pros: Wide variety of tools available, support for .NET framework with more detailed server side controls (good for hosting game on multi-platform)  Cons: Windows has a lot of licensing fees, more expensive than linux, can consume more resources on average over linux. | Pros: Can use devices features (location services for competing locally?) for faster performance, in-app purchases offer marketing opportunity  Cons: Must create platform specific (ios, android) application, two separate code bases = more to keep up with, must make it through approval process of app store. |
| **Client Side** | Licensing fees for Apple developer program  Platform specific UI design means more time spent coding to reach multiple platforms. | Open-source tools/library means much cheaper development. C++ or Python proficiency and knowing Linux system admin is important. | Can be cost effective depending on which tools you utilize.  Large community means faster development.  C#, UWP or .Net expertise | Fees for each app store. Specific UI design depending on which app store you’re expanding to.  Swift expertise and knowing mobile UI/UX guidelines is essential. |
| **Development Tools** | Node.js or Django (web server)  Swift, javascript, python all widely prevalent.  MacOS UI and  Xcode (for swift/obj-c)  Interface builder  MySQL for DBMS | Apache (web server)  MySQL or MongoDB for data management.  Node.js is popular for webapps (javascript)  Visual studio is a popular IDE. | ASP.NET or Ruby on Rails for web development.  Visual Studio popular IDE.  My SQL or SQL server for DBMS.  Java, python and C# all common languages. | Swift for IOS and Java for android are both popular languages for mobile app development.  React Native let’s you code once and then helps translate for ios and android. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.> After reviewing the data gathered, using windows as a base operating system would be most cost effective in the long run. Since the client is already looking to expand for the first time it is easy to assume they will want to expand again in the future, and it will be easier to migrate to windows than it will be to go to Linux. Windows is widely used, easily accessible, and has many tools available for development.
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.> Windows architecture is a two-component system consisting of user mode and kernel mode. A lot of the applications and user software are executed in user mode, it also prevents users from accessing and altering important system data/resources. Kernel mode is what handles drivers, data management, RAM, and system processes. By separating these functions Windows can run more efficiently.
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.> Cloud storage is currently the most efficient solution to the client’s storage management needs. It can be easily integrated to run within windows and helps avoid the costs of setting up one’s own hardware. It’s easily expanded for future needs and there are many different plans available that help meet a growing business’s requirements. Amazon S3 is currently one of the more cost-effective cloud storage options.
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.> Through a combination of paging and use of physical and virtual address space Windows can efficiently handle a high load of memory. Paging frees up RAM by temporarily storing data to the disk (not permanently writing to the disk) for easy access when needed.
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).> A RESTful API allows for efficient communication across multiple platforms. Storing information/data on a server that just needs to be accessed on the client side allows for the software to load a player’s data from different systems (cross-platform). Windows client-side is affordable, has a large library of tools available, and is easy to use.
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.> Many of the server start up software available today has security features built in. Utilizing extra steps for users like two factor authentication helps to prevent hacking or misuse of a player’s information. Routine updates to security features help to upkeep the effectiveness of the security measures in place. Having a security audit done annually to check the system would be a good thing to have on the maintenance list. Also, the client could utilize firewalls or VPNS to protect user data between systems.