

**Draw It or Lose It**

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 04/15/22 | Zachary Mohler | Added recommendations section. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has requested assistance in developing a new web-based game, “Draw It or Lose It” from their existing Android app game.

## [Design Constraints](#_2et92p0)

To operate in a web-based environment, the game must:

* Be lightweight as to not overload servers
* Be simple enough to run average systems
* Allow online play

## [Domain Model](#_8h2ehzxfam4o)

Below is the UML Diagram for the gaming room. On the upper left is the program driver class. This is used to run the overall program. The program driver has an associated class called SingletonTester, which is used for development purposes.

On the upper right is a class called Entity. Entity is an abstract class used as a baseline for its child classes: Game, Team, and Player. These child classes inherit the base properties from Entity and add their own properties.

Game adds a *teams* variable to store each team’s data. Team adds a *players* variable to keep track of a team’s platers. It should be noted that the Game class has a zero-to-many relationship with the Team class, and the Team class has a zero-to-many relationship with the Player class.

Finally, on the bottom left is the GameService class, which keeps track of all running games, players and teams.

**"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** | Relatively simple administration. Works well for different client platforms. Becomes more difficult as scale increases. Moderately expensive. | Free to use. Low priority target for attacks. Not strenuous on hardware. Very limited available professional programs. | High compatibility. Easy to use. Well maintained with guaranteed support. Highly vulnerable to attack. Expensive. Very hardware intensive. | Relies on client hardware. Relatively low cost. Very poor scalability |
| **Client Side** | Mac sometimes does things slightly different than other platforms which may cause a difficulty in implementation. Very in-depth knowledge of the system is required. | High compatibility, as Linux is very efficient at compiling code for multiple platforms. Easy environment to develop for. Properly applying compatibility may take time. | Very common client platform. Well documented. Requires in-depth knowledge to properly secure programs which may result in high cost and development time. | Lowest cost in terms of development time. Easiest client type to support. Limited hardware capabilities. Highly varied hardware across multiple clients. |
| **Development Tools** | UNIX based works great for development for linux servers. IDEs typically easier to learn starting out. As application complexity increases, performance tends to decrease. | Overall higher performance. Open-source standard increases learning curve and margin of error but creates more flexibility. | Very robust and well documented software. Many available third-party tools. High licensing cost. | Development for mobile devices is by far easiest. Cheaper licensing. Lower capability and ceilings. |

## 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 recommend a Linux operating platform for this project. A Linux operating platform will provide a great balance of budget and security that will greatly benefit a project of this type. Furthermore, Linux also is a very adaptable platform which will add the benefit of compatibility.
2. **Operating Systems Architectures**: A multi-tiered architecture is the absolute best option for this project. In terms of performance, a multi-tiered architecture will run much faster, which is very important for a time-based game such as *Draw It or Lose It*. A multi-tiered architecture is also more secure than a client-server architecture as it doesn’t allow for direct user interaction with the database. Overall, a multi-tiered architecture is a very good fit for a web-based game like the one proposed in this project.
3. **Storage Management**: I recommend at least 1TB of HDD storage for user data alone. This will provide more than sufficient storage space for user information but can also be easily scaled up. I would also recommend storing game files on an SSD for the increased performance. The game files will be a constant size, so the cost can be minimized by using the smallest SSD available to fit the final size of the game.
4. **Memory Management**: Properly assessing the amount of memory needed for this type of project is difficult to do without any rough estimate of the number of users who will be playing at any given time. That being said, I recommend 16GB memory for the system. This will likely provide sufficient memory due to the relatively small size of the game, providing plenty of memory to be left for support of many concurrent users. If the expected number of users at any given time is expected to be in the multiple thousands though, I would recommend increasing the amount of memory.
5. **Distributed Systems and Networks**: While a REST framework may not be exactly applicable in the scope of the game itself—it may be beneficial to implement as a way for users to view things like leaderboards and other user’s information. This would add another layer of complexity though, and if a multi-tiered architecture is used, I cannot recommend it.

I do recommend a queueing system for the game. This will add a layer of protection from overstraining the system, preventing crashes or slowing.

1. **Security**: The Linux operating platform provides a very good base for security. By default, the permissions structure alone provides a decent layer of security. This along with the fact that Linux is a lower priority for attackers provides a great base to build well made, secure programs on. Best practices should still always be used though, of course.