

Draw It or Lose It

# **CS 230 Project Software Design**

Version 3.0

## 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 | 09/16/2023 | Trenton Grossheim | The Initial formation of this document |
| 2.0 | 09/27/2023 | Trenton Grossheim | Added the Evaluation section. |
| 3.0 | 10/13/2023 | Trenton Grossheim | Added the recommendations section |

**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 wants to be able to play Draw It or Lose it on the web, but Draw It or Lose it is only an android app. The purpose of this project is to make Draw It or Lose for the web, that works on multiple operating systems.

## Requirements

* Ability to create games and assign them teams.
* Ability to create teams and assign them players.
* Ability to create players.
* Allow the Drawing player to render an image.
* Allow the guessing player to guess the answer.

## [Design Constraints](#_2et92p0)

* Must be written in a language that supports making web applications or that has libraries that do.
* Must be able to support image files, with either built-in or 3rd party libraries.
* Must be designed so that multiple computers can connect to one game at a time.

## [Domain Model](#_8h2ehzxfam4o)

I will start at the top left and move to the right. First off, we have the ProgramDriver Class which is the class that holds the main function, nothing more. The SingeltonTester classes test if our GameService class is indeed using a Singleton design pattern, it is being used in the main function. Then we have the Entity class which is the superclass of the Game, Team, and Player class. The GameSerive class has zero-many games, the Game class has zero-many teams, and the Team class has zero-many 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](file:///C:\Users\Guest1\Downloads\CS%20230%20Project%20Software%20Design%20Template.docx#_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 must 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 is good if you are going to be making features for mac clients. Requires mac hardware which is usually expensive. There is no user license, but there is not a lot of third-party content. | Linux are one the most tried and true server operating platforms, being one of the oldest and still in use today. Linux machines have high security and are open source. | Windows Servers has good interface and has lots of third-party content. But you do need a user license and it is prone to getting viruses. | Mobile devices are not going to be powerful enough to do anything but rudimentary features. |
| **Client Side** | Desktop version of the client is going to be on a larger display and typically won’t need to touch screen support. | Desktop version of the client is going to be on a larger display and typically won’t need to touch screen support. | Desktop version of the client is going to be on a larger display and typically won’t need to touch screen support. | When making web apps for mobile, one must develop the app to fit onto a small screen and be able to be used with touch screen features. |
| **Development Tools** | The MEAN stack is a stack that uses tools that are all based on JavaScript. It can be used for any desktop development. | The LAMP stack is one of the most used stacks. It is based on a Linux operating platform with Apache, MySQL and PHP. | The MEAN stack is a stack that uses tools that are all based on JavaScript. It can be used for any desktop development. | Mobile can use the same as desktop stacks, it is only about the implementation of those tools. |

## 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**: The server should be hosted on a Linux server so we can have full customization but no user fees, but the client side should be developed with HTML and JavaScript, so it can run on any desktop operating system that can run a web browser.
2. **Operating Systems Architectures**: The Linux server will have a typical Linux Architecture. Linux machines can be viewed as a hierarchy of layers starting with the hardware, then the kernel, shell, and finally the application layer.
3. **Storage Management**: At least one solid-state disk storage, with as much storage as reasonable, the smallest one with 256 GB would be sufficient for our project. It would be better to get two so we can replace it when it retires. It would also be smart to get a magnetic tape drive for back-up storage.
4. **Memory Management:** We won’t need a huge amount of ram for our small web application, but you can always have more ram. We can make sure that we never run out of ram by only loading the assets being used and assets that have links on the current page.
5. **Distributed Systems and Networks**: For the application to be able to run on any client-side web browser it will have to be written in html, CSS, and JavaScript. We can use HTTPS to send REST API between information between the server and client.
6. **Security**: HTTPS uses encryption keys that can be used to verify users and make sure they are verified throughout the website. We’ll be using a REST API so the client side will not directly be able to access the server side, preventing malicious behavior from the client side.