

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 09/20/22 | Janai Cano | First version of Software Design outlining design constraints, system architecture view, domain model, evaluation, and recommendations. |
| 2.0 | 10/01/22 | Janai Cano | Edited the executive summary and design constraints, added my evaluation for server and client side, and added Evaluation table. |
| 3.0 | 10/14/2022 | Janai Cano | Edited evaluation table, added recommendation |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is currently available as an Android App. The game is played by two teams who are given 30 seconds each to guess what a drawing is. The drawing is a stock image drawing that is shown in pieces to each team. The first team to guess what the picture is, wins! Our client needs us to create a web-based version of their game. The game should have the ability to host two teams per game, with unlimited number of players per each team. Every game, team, and player must have unique names. The stock images used for each game needs to be securely stored, and player information also needs to be stored and securely kept. Following is my analysis of which API should be implemented to create an environment that fits these criteria for Draw It or Lose It.

## [Design Constraints](#_2et92p0)

The web-based version of this game will be designed based off Android OS, so we may need to implement a different programming language than the original game to ensure that Draw It or Lose It has all features functional on all three major computer OS’s (Windows, Mac, and Linux) and most web browsers that these OS’s implement. Each instance of the game must be unique. Player names and team names also must be unique. User information needs to be stored securely, along with stock images used by the game. Where the stock images are being pulled from/ stored is a constraint. We want performance to be optimal and images to be rendered in real-time to each player. Memory allocation will be key. The client has not specified server preference, whether we will be using cloud-based servers or using on-site servers. As of now, there is no limit to number of players that can be allowed into one game instance, so memory-allocation and space for player information must be factored into the architecture design. The game uses stock images that should be stored in a server and presented in real-time, piece by piece to the individual players. Evaluating how the Android app is set up may be useful if the client decides they want an iOS game in the future.

## [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)

Below is the visual for the UML. The ProgramDriver class contains the main() method which employs the SingletonTester to test if there is already an instance of the game found in Game class. Entity is a parent class to all four classes: GameService, Game, Team, and Player. These three classes inherit all of Entity class attributes and also implement polymorphism so that each have unique attributes of their own. There can be many players on a team, and many teams in one game, but only one game instance. GameService should be the class that ensures that this happens using the singleton method.

**"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.**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Can use cloud-based servers or local server hosting with additional software installed. Licensing cost are high to use Macs as servers. | Can use cloud-based servers or local server hosting. Local hosting on Linux is preferred, as there are no licensing costs unless going with cloud-based hosting. | Can use cloud-based servers or local server hosting. Local hosting on Windows is preferred, as licensing cost is low and easy to use. Additional cost for cloud-based hosting. | Android – already available to users using clients’ current servers.  iPhone – cannot be used as a server. |
| **Client Side** | Safari-native browser, but most web browsers are compatible with Mac as of 2022, including Edge, Chrome, and Opera. | Preferred web browser must be installed, but compatible with most as of 2022, including Chrome, Edge, and Opera. Safari is non-compatible. | Microsoft Edge is native browser, but most web browsers are compatible as of 2022, including Safari, Opera, and Chrome. | Android – compatible with most browsers, including Safari, Chrome, and Edge.  iPhone – Safari native browser compatible, also can use other web browsers including Chrome, Opera, and Microsoft Edge. |
| **Development Tools** | iCode for Swift development, but as of now, many IDE’s are Mac compatible. HTML, CSS, JavaScript, Pytho, PHP, and Ruby can all be access and used. Multiple development teams may be used. Can be tested cross-platform. Licensing costs may apply. | Python comes installed on Linux systems, but is slow for execution. If using another scripting language, like Ruby or Perl, we could use Ruby on Rails or GitHub. Multiple development teams may be used. Can be cross-platform. Licensing costs may apply. | Visual Studios comes with Windows and is a nice IDE. Supports HTML, CSS, JavaScript, Python, Java, and others. Multiple development teams may be used. Can be cross-platform. Licensing costs may apply. | Cannot develop on any mobile platform as of 2022. Testing can be done on mobile web browsers, but no IDE’s exist for development. |

## [Evaluation](#_2o15spng8stw)

## 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 Windows as our Operating Platform to develop this web application further. It is compatible with most web browsers, including the Mac-native web browser, Safari.
2. **Operating Systems Architectures**: Windows comes in 32 and 64-bit architectures. It allows us to read, write, and edit files while also managing user access to certain files. It handles memory using virtual memory multitasking. Using the OSI model would serve our client-server purpose.
3. **Storage Management**: File-storage system management, including a file-tree database to store images, user files, game data, and styling.
4. **Memory Management**: Using a 64-bit Windows system will have optimal RAM for designing this web application. We can add more if need be in the future, it is upgradable on Windows systems.
5. **Distributed Systems and Networks**: Using the OSI model of Windows, the client will request information from the server, the server will take that request, get the data from either the hard drive (storage) or RAM (memory), and return it to the client. This is a kernel-based model, so memory is optimally allocated.
6. **Security**: password implementation and encrypted files will be key in our application design. Users will be given restricted security authorization, needing passwords to access their game data, user settings, and other user profile information. Other user information will only be accessible through permissions. Creating users and administrators will be key in our design.