

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 | 03/23/24 | Shayne Greene | Initial release |

**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 is looking to develop a web-based game based on their existing game, Draw It or Lose It. Currently Draw It or Lose It is only available on Android. Once the game is developed, significant testing will be needed to assure compatibility across several of the popular browsers.

## Requirements

* Game needs to be web-based,
* Staff needs help in streamlining development,
* Game will have one or more teams with multiple players involved,
* Game will allow only unique team names,
* Only one instance per game will exist in memory at a time.
* Database for stock drawings used as in game clues.

## [Design Constraints](#_2et92p0)

* For a web-based game, development should be done in Java or JavaScript.
* Training for staff once the game has been developed for maintenance and new features.
* A database is needed to hold images used in-game.
* Allowing multiple teams for each game.
* Each team has multiple unique players.
* Following the current game rules

## [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)

In the UML model below, we can see that the Game, Team and Player class inherits from the Entity class. The GameService class can have zero-to-many Games, Games can have zero-to-many Teams, and Teams can have zero-to-many Players. The ProgramDriver classes uses a SingletonTester class.

As shown in the UML diagram below, these are some of the object-oriented programming principles utilized:

Inheritance – The Game, Team and Player classes inherit fields and methods from the parent class, Entity. Because each of these objects, once created, will have an id and a name. And each object will share the parents methods, getId() and getName().

Encapsulation – When an object is created, the only way to access fields are through accessors. To unsure that the Game, Team or Player object instance fields are not updated there are no mutators. The only way to set the data is through initialization.

For instance, when a player is added, a unique ID number is give to the player. There is no way to set the name to that player object, except when the Player object is created.

Polymorphism - The child classes inherit the parent’s method toString. But each of the child classes has their own toString method with the same syntax that will produce a different output.

**"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** | **Advantages**:  • Performance  • Improved security.  • Familiar GUI  **Disadvantage:**  • lack of Mac-specific software options. | Advantages:  • lower cost than Mac or Windows.  • opensource or an enterprise distro (like Red Hat)  There is a web-based hardware monitoring tool such as Webmin.  Disadvantage  • Lack of an interface.  •Setup and updates will be through a command line. | **Advantages**:  •Simple GUI environment makes it easy to access tools like adding users and other AD functions.  **Disadvantage:**  • More vulnerable for attacks  • Higher licensing costs | Can be done using tools like Termux to emulate a Linux environment. But this not recommended for hosting in this case. |
| **Client Side** | Using HTML5 for the web UI, you will cut costs for testing the across the top browsers.  Users will likely already be using one of these, Safari, Chrome, Firefox, Brave. | If the client is running an Ubuntu distro with a GUI, Using HTML5 will ensure cross browser usability. Firefox seems to be the preferred browser for Ubuntu users. | Using HTML5 for the web UI, you will cut costs for testing the across the top browsers.  Users will likely already be using one of these, Edge, Chrome, Firefox, Brave. | Development for a mobile device, screen real estate is a factor. There will be a cost to develop a version that will be used via a mobile device. Size of photos, playability. For instance, if buttons are too small, it will become frustrating to play. |
| **Development Tools** | **IDE options:** NetBeans  Eclipse  Atom  Xcode  Sequel Pro.  **Programming languages:**  iOS  Swift  Android  Java  Kotlin  GO  Cross platform  Python  C#  Flutter  JavaScript  Node.js | **DE options:** NetBeans  Eclipse  Atom  Code::Blocks  Android Studio  **Programming languages:**  iOS  Swift  Android  Java  Kotlin  GO  Cross platform  Python  C#  Flutter  JavaScript  Node.js | **IDE options:** NetBeans  Eclipse  Atom  Android Studio  **Programming languages:**  iOS  Swift  Android  Java  Kotlin  GO  Cross platform  Python  C#  Flutter  JavaScript  Node.js | There may be development tools that are available for mobile devices, but it is not an advised method of development. |

## 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**: Window would be the choice for the backend support. Where the upfront licensing cost would be higher than that of Linux, support and the user friendly UI would be beneficial in the long run.
2. **Operating Systems Architectures**: Using VMware, you can spin up several working backend servers that users will connect to. To scale up, you would just need to create a new instance of a virtual machine.
3. **Storage Management**: Using a SQL server for data storage. For example: user profiles, player / team stats will be stored in this database. This database will be shared across all backend servers
4. **Memory Management**: Once a game is instantiated, all the images will be cached on that server and shared across each game instance that is added to that server. The same would go for the dependencies for the source code.
5. **Distributed Systems and Networks**: Clients will access the game via their favorite web browser. Once the user is authenticated, they will be sent to one of the servers from a server pool using a load balanced server. This will allow for an even distribution of users over a range of servers to spread the workload. If one of the servers goes down, it will be taken out of the pool and users will be routed to one of the working servers.
6. **Security**: Users will require to sign up or sign in with an account. The password strength requirement should requirement should be complex. Consider dual factor authentication for developers that have access to sensitive information and databases. A service such as DUO is platform independent and will work across any OS. Guest accounts will be allowed, but with limited access.