

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

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 | 11/08/2022 | Heath Banak | Update software design elements |
| 1.0 | 11/13/2022 | Heath Banak | Update software design elements |
| 1.0 | 11/26/2022 | Heath Banak | Add design constraints |
| 1.0 | 11/27/2022 | Heath Banak | Expand recommendations |
| 1.0 | 12/11/2022 | Heath Banak | Update recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming room wants to create a web-based version of their Android-only app Draw It or Lose It and does not know how to set up the environment. The game must serve multiple platforms. The app must support gameplay with multiple games that each have multiple teams with multiple players; game names and team names must be unique. The game contains a library of stock drawings that are fully rendered in 30-seconds and each game comprises four one-minute rounds; if the team fails to guess the puzzle in time, the remaining teams have 15 seconds to guess.

## [Design Constraints](#_2et92p0)

Web browser

* The game will be deployed via a web server and offered to clients through web browsers
* The game should run on the most popular browsers: Chrome, Safari, Firefox, and Edge; browser features and performance create development constraints.

Storage and memory

* The image library must be large enough to support the game long-term
* The large image library must have adequate storage and memory for gameplay

Security

* The image library must be secure from outside users
* Authentication procedures and role-based access will be used
* Personal information of users must be secure.

Gameplay

* Each game must have a unique name and id and support multiple teams.
* Each team must have a unique name and id and support multiple players.
* Each player must have a unique id.
* One instance of the game can exist at a time.

Development and performance

* Clients OS: MacOS, Linux, Windows, Mobile
* The programming languages used may vary depending on the browser being targeted
* Users must experience gameplay in real-time; the server must be fast and able to accommodate many clients

## [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 entity class acts as a superclass to the Game class, Team class, and Player class. The Entity class encapsulates attributes and methods common to the subclasses. The child classes inherit these methods and attributes and can access them without having them explicitly defined. The toString() method is overridden in each child class, showing the OOP property of polymorphism.

The GameService class follows a singleton design, so only instance can exist at a time. This allows for no games to be duplicated. The GameService class can contain 0 to many instances of the Game object, and the uniquely named Games are stored in an ArrayList. Similarly, the Game class can contain 0 to many instances of the Team object, and the uniquely named Teams are stored in an ArrayList. Following this pattern, the Team class can contain 0 to many instances of the Player object, and the Players are stored in an ArrayList. The GameService class also holds the unique ID for the next Game, Team, and Player, which can be retrieved by the classes of the same names. A Game is initiated with the addGame() method of the GameService class.

Finally there is the ProgramDriver class which contains the main() method and drives the program. This class controls program execution and instantiates the other classes. The ProgramDriver class uses the SingletonTester to ensure that no more than one instance of the GameService object can be created.

**"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** | Website developed with MacOS runs on any OS  -------------------------  Pros  Hosting environments are user-friendly  High security  Integration with Xcode  -------------------------  Cons  Hosts can be difficult to find  Expensive | Website developed with Linux runs on any OS  ------------------------  Pros  High security  Open-source OS is configurable  Lowest cost  ------------------------  Cons  Requires more expertise than other options | Website developed with Windows runs on any OS  ------------------------  Pros  Popular with programmers  ------------------------  Cons  Medium security  Similar cost to Mac | Website developed for mobile devices will run on mobile browsers  No server necessary  iOS apps can be uploaded to App Store, Android apps to Play Store  -------------------------  Pros  No hosting necessary  -------------------------  Cons  App store has approval process plus fees  Play store has fees |
| **Client Side** | Most developers are familiar  Fast development, game only needs converting to new platforms  Supported browsers:  Safari  Chrome  Firefox | Most developers are familiar  Fast development, game only needs converting to new platforms  Supported browsers:  Edge  Chrome  Firefox | Most developers are familiar  Fast development, game only needs converting to new platforms  Supported browsers:  Edge  Chrome  Firefox | Native app:  Development is more niche  iOS apps must be approved for the App store  Mobile web-based app:  Developers are familiar  Fast development, game only needs converting to new platforms  Supported browsers:  Safari  Chrome  Firefox |
| **Development Tools** | Frontend:  HTML  CSS  JavaScript  Backend:  PHP  Java  IDEs:  Xcode  VSCode  WebStorm  IntelliJ (License)  Mobile development:  Browser Developer Tools can be used for responsive layout development | Frontend:  HTML  CSS  JavaScript  Backend:  PHP  Java  IDEs:  Xcode  VSCode  WebStorm  IntelliJ (License)  Mobile development:  Browser Developer Tools can be used for responsive layout development | Frontend:  HTML  CSS  JavaScript  Backend:  PHP  Java  IDEs:  Xcode  VSCode  WebStorm  IntelliJ (License)  Mobile development:  Browser Developer Tools can be used for responsive layout development | Languages:  Swift  Objective-C  JavaScript  IDEs:  Xcode  VSCode  Atom  WebStorm  Development is done on non-mobile platforms. Desktops or laptops will be required |

## 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**:

Linux is the recommend operating platform due its high level of security, low cost, and accessibility. Linux offers the necessary role-based access control through controlling privileges. Unlike systems like Windows, Linux isolates the root privileges rather than giving administrator access by default upon setting up the operating system. Linux also has the lowest overhead concerning hardware. Finally, Linux is free and open source, meaning core changes may be made to the system if needed.

1. **Operating Systems Architectures**:

The Linux kernel contains only core components, with separate process implemented outside of the kernel. This uses less disk space and eliminates the need for context switches. Linux also executes functions through system libraries that do not have direct access to the kernel, providing additional security.

1. **Storage Management**:

For storage, Draw It or Lose It has the options of local storage or cloud storage. One benefit of a serverless architecture deployed on the cloud is price. Cloud storage is priced by execution rather than a fixed price for a fixed amount of space. This structure allows the program to scale and descale as necessary without a fear of wasting storage that has already been paid for. Serverless is also beneficial for developing in multiple environments, as well as offering faster accessing of data remotely. One downside of the seamless scaling offered by the cloud is a lack of control. This lack of control stems partially from the trust required from the third-party cloud host. A local storage device allows for total development control, however, also requires much more manpower and funds. Serverless functions also require private APIs, while local storage does not.

1. **Memory Management**:

Linux uses virtual memory and demand paging, which decreases the physical memory size and allows for multiple programs to run concurrently. Using partial memory also loads and runs the app faster, improving user experience. Compared to Mac and Windows, Linux requires very little RAM, leaving the greatest amount of RAM left for the rest of the system. Draw It or Lose It will load images into Ram and slowly render them for the user during the game.

Draw It or Lose It will require specific approaches to ensure the memory usage is managed effectively. One memory consideration is the storage of the 200 images to be used in the game. Another is the storage of user information and game information. Finally, there will be user input that must be handled. A solution to this problem is to implement a virtual memory. A virtual memory allows a program to run that is larger than the physical memory. The program can store data not being actively used in the virtual memory and send data to main memory as needed. When the Draw It or Lose It program runs, the photo being rendered must be stored in main memory. However, the images not being used may not all require main memory storage. If, for example, 50% of the images were stored in virtual memory, the remaining 50% would be available for immediate gameplay. Reducing the amount of data in main memory by storing it in virtual memory will benefit the capability of the program.

1. **Distributed Systems and Networks**:

Draw It or Lose it may be deployed in a cloud-based, serverless architecture. Multiple development environments can be deployed easily, there is no concern for overloading local servers, and the game has the potential to reach a greater audience.

Draw It or Lose It will use REST API exchanges between clients and servers over HTTP that are lightweight and stateless. Using HTTP, session data can be uniquely identified, and messages from clients to the server represent the same context to all Oss.

1. **Security**:

One aspect of the Linux security features is process independence. Each system process has its own virtual address space that is independent of other processes. Linux also requires user authentication, with the least number of privileges enforced by default. Together, these limit the scope of user access and increase security.

Linux member access is one of its key security features. Files are protected in Linux with three layers of permissions based on user and role. Multiple checks must be programmed to see whether the user is, for example, an admin or a regular user. This aids in security because users logged in as users cannot access privileged files or data that a user with an admin role could.

Another security measure to be implemented is read-only access given to all users. If files must be accessed by a user or by the file owner, permissions are able to be changed. This file protection is done most easily on Linux.

A serverless architecture allows for less security, as many of the security responsibilities are placed on the third-party cloud provider.