

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 | 05/22/2022 | Ryan Kittelson | Initial version of The Gaming Room Software Design Documents. Implemented executive summary, |
| 2.0 | 06/01/2022 | Ryan Kittelson | Added Client Side, Server Side, and Development tools |
| 3.0 | 06/18/2022 | Ryan Kittelson | Added recommendations |

**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)

My client the Gaming Room wants to develop a web-based game based that serves multiple platforms based on their current game, Draw It or Lose It. That is currently only available for Android.

## [Design Constraints](#_2et92p0)

* Technical Constraints
  + Software must run on a web-based environment to allow for multiple platforms to use it.
    - Based off existing game, Draw It or Lose It
  + Large library of stock drawings needs to be available.
  + Application will render images on the screen while teams have 1 minute to guess what is being displayed
  + Software is currently available only for Android
  + oud hosting of the training videos
  + A game will have the ability to have one or more teams involved.
  + Each team will have multiple players assigned to it.
  + Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
  + Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.
* Business Constraints
  + Do we have developers who can help to maintain the software?
  + Cost involved for maintaining new web-based design.
  + Hosting fees
    - What are the costs for web hosting for the server?

## [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)

This UML Diagram shows the Entity class being the parent class for 3 three different child classes listed as Game, Team, and Player. All three of these classes use inheritance to capture everything from the Entity class. Since Entity is a parent class, and the diagram shows the constructor is private, we must write it as a protected class. Abstraction is shown because in each class there is a private list contained. GameService uses a singleton pattern to make sure that there is only ever one to exist. There can be multiple games to exist at once as well as multiple teams and multiple 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](#_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** | * UNIX-like server OS * Builds on top of macOS * Fairly uncommon OS * Great if Mac devices on network * Easy administration * Intuitive GUI * Great support * Only runs on Apple Hardware so difficult for hosting a web-based software application for multiple devices * Would avoid this in this scenario | * Roughly 80% of all servers use some variation of Linux * Open Source * Very high security * Free * Wide variety of distros * Ability to control the system using a GUI * Server OS administered entirely from command line * Requires more technical knowledge than other operating systems * Lack of long-term support for some distros and updates | * Dedicated server version * ARM processor architecture * Roughly 20% of market share. * Requires a paid license * Virtual memory management * Intuitive GUI * Support for symmetric multi-processor systems * Third-party application support * Many versions * More virus threats than other platforms. * Closed source | * Lifespan of mobile devices much smaller than desktops * Mobile device servers can be done, but they are not very reliable * Web apps on mobile devices are slower than native apps * Will update automatically * Web apps work on multiple devices as they run within the browser itself * Old Android devices have been converted to dedicated servers |
| **Client Side** | * Developers need expertise in language decided on * Consistent internet connection * Apple development costs are higher due to higher cost of goods sold * Testing on different browsers * Hardware requirements are expensive for Mac * Testing on multiple machines * Latency | * Developers need expertise in language decided on * Consistent internet connection * Testing on different browsers * Memory usage * Content distribution network can help with improvement of speed * Latency | * Developers need expertise in language decided on * Consistent internet connection * Use the SOLID principles of programming: * Single responsibility * Open-closed * Liskov substitution * Interface segregation * Dependency Inversion * App meats Certification standards * Testing in different browsers * Security issues can occur easier in Windows * Latency | * Image optimization * Compression of code * Testing in different browsers * Scalability * If creating a native app, experts in Java and Swift * Updates are done within web app, so all devices are updated at once * Cost efficiency in web app vs native app * Hardware limitations on web-based apps * Consistent internet connection * Latency |
| **Development Tools** | * Python * Java * Swift * Objective-C * Eclipse * Visual Studio Code * Xcode * Maccy * Amphetamine * Rectangle * GIMP | * Python * C * C# * C++ * Perl * Java * Eclipse * Visual Studio Code * Vim * Ubuntu * JavaScript * Atom * ReactJS * Bootstrap * API Testing Tools * NPM (default package manager) | * Python * C# * C++ * Visual Studio Code * Perl * Java * Front-end developers work with HTML, CSS and JavaScript * Eclipse | * Android native apps are built in Java using Eclipse IDE * Apple native apps are built using Objective-C or Swift * Since this is a web app, we will be using JavaScript, HTML, CSS, and Python * jQuery * Sencha Touch * React Native JS can produce cross-platform development creating both web app and native app |

## 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**: For Draw It or Lose It, I recommend the Linux operating platform for designing the game. Linux is the most common operating platform for the server side, and since our game will be utilized as a client-server system with multiple devices accessing it, we need to make sure that our server side is robust. Linux is open-sourced and very secure which will help keep our important data secure. We will setup our central server to make everything in the game happen, and the users will simply send inputs to the server. I recommend using the Ubuntu distro for this as well due to its ease of use and support for multiple devices. I chose Linux over Mac and Windows because of its stability, security, and flexibility.
2. **Operating Systems Architectures**: The various components of the Linux operating system’s architecture are the kernel, system library, system, hardware layer, and shell utility. The kernel is one of the core aspects of the OS. The kernel is responsible for the major functions in the Linux operating system. The system libraries are important for special functions and are applied when needed. System utility programs are also involved for doing individual activities. The hardware layer is where the physical hardware comes into play. Peripheral devices such as the CPU, SSD, and RAM are examples of the hardware layer. The shell is an interface between the kernel and the user. It allows users to run commands to run different functions of the kernel. We will utilize a client-server architecture for this system because it will allow us to keep our important data on the server and allow the users to simply send inputs to the server.
3. **Storage Management**: To improve speed of accessing information, we will utilize solid state drives instead of hard disk drives for this project. Since the size of our images will not be changing, the standard partitioning method should be used over the LVM method. Since our storage isn’t too crazy in terms of size, we will utilize the ext4 file system or the extended file system due to its stability. Assuming the 200 HD images equal roughly 1600 megabytes, we are going to want to require at least 16GB or RAM for the clients to handle the client-server application. 128GB SSD should be sufficient for maintaining the image files.
4. **Memory Management**: One recommendation for memory management for this system will be to utilize virtual memory and swap pages to make the system appear to have more memory than it has and to free available memory. Memory mapping is something that we can use for our images to be stored. This will allow us to link the contents of the files directly to the virtual address spaces. By allowing the use of virtual memory, each program can reserve a section of memory making it so no other application can reserve it. Another tool that will be helpful in our memory management is the Translation Lookaside Buffers (TLB). This helps the system know where everything within the virtual memory lies. To handle an influx of user requests, we can also utilize page caching techniques to speed up response times.
5. **Distributed Systems and Networks**: Since we are running this as distributed software, using a REST API can help us with privilege management since most of our game will be ran on the central server. The REST API will be installed on the central server to communicate effectively with the client devices. Distributed systems do a great job of spreading out requests and workload so that a lot of requests can be supported at once.
6. **Security**: By using the principle of least privilege we can ensure that all users are only granted the right level of permission to perform the tasks they need to perform. By keeping the users separated from most of the data, the application becomes much easier to maintain from a security perspective. Linux has a very easy to use permission management system for restricting access. Linux also has a lot of built-in security features that make it a great choice for this application. By requiring users to create an account to access the system, we can add another layer of security for the user’s data as well as access to the game.

References:

*6 considerations for Windows Universal Application Development and phone development*. Clarity. (n.d.). Retrieved June 6, 2022, from https://www.clarity-ventures.com/articles/6-considerations-for-windows-universal-application-development-and-phone-develo

Abhinav, K., Admin, Robert, & Xmc.pl. (2020, September 10). *Tutorial: Beginners guide on linux memory management*. GoLinuxCloud. Retrieved June 19, 2022, from https://www.golinuxcloud.com/tutorial-linux-memory-management-overview/#What\_is\_memory\_over-allocation\_over-commitment\_and\_OOM

Agrimbau, T. (2014, March 11). *Developing mobile web applications: When, why, and how*. Toptal Engineering Blog. Retrieved June 6, 2022, from https://www.toptal.com/android/developing-mobile-web-apps-when-why-and-how

*Architecture of linux - javatpoint*. www.javatpoint.com. (n.d.). Retrieved June 18, 2022, from <https://www.javatpoint.com/architecture-of-linux>

*Chapter 3 memory management*. The Linux Documentation Project. (n.d.). Retrieved June 19, 2022, from https://tldp.org/LDP/tlk/mm/memory.html

David BoltonMarch 9, 20157 min read. (2018, May 25). *Best programming languages for linux DEVS*. Dice Insights. Retrieved June 6, 2022, from https://insights.dice.com/2015/03/09/best-programming-languages-linux-devs/

Garn, D. M. (2022, May 2). *How do windows, linux, and macos compare?* Default. Retrieved June 3, 2022, from https://www.comptia.org/blog/os-comparison

*Memory management in operating system*. GeeksforGeeks. (2021, November 22). Retrieved June 12, 2022, from <https://www.geeksforgeeks.org/memory-management-in-operating-system/>

*REST API for privilege management for unix and linux*. BeyondTrust. (n.d.). Retrieved June 19, 2022, from https://www.beyondtrust.com/docs/privilege-management/unix-linux/admin/overview/rest-api-pmul.htm#:~:text=The%20REST%20API%20can%20be,1.0%20and%20later.

*Server operating system: Server Os Types & How to choose*. Knowledge Base by phoenixNAP. (2022, March 10). Retrieved June 4, 2022, from https://phoenixnap.com/kb/server-operating-system

Stevens, E., England, E. S. O. from, Stevens, E., & England, O. from. (2021, September 7). *18 pros and cons of mobile apps vs. web apps*. CareerFoundry. Retrieved June 6, 2022, from https://careerfoundry.com/en/blog/web-development/what-is-the-difference-between-a-mobile-app-and-a-web-app/

Tirthakar, S. (1969, January 1). Explore the best linux tools for web developer. Retrieved June 6, 2022, from https://linuxhint.com/best-web-development-tools-linux/