

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 3 /20/22 | Allen White | Initial information setting |
| 2.0 | 4/24/22 | Allen White | Completion of more detailed information and recommendation |

**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 client (The Gaming Room) wants to extend their Android-based game, “Draw It or Lose It”, into a web-based application which can serve multiple platforms.

The program will render images for a library of stock drawings. The game consists of four rounds of play lasting one minute each. The drawings are rendered steadily until completion at 30 seconds. If the designated team does not guess the answer to the puzzle before 30 seconds, the other teams have the chance to guess once to solve the puzzle (15 second limit).

## [Design Constraints](#_2et92p0)

* Hardware platform – Serverless
* 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.

## [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)

All classes in the program exist in the gamingroom.com namespace as seen by “com.gamingroom”, at the top of the diagram.

The Entity class is the parent (is inherited by) the Game, Team, and Player classes. Each of the children inherit methods and properties from Entity, and each add their own specific public and private members and functions. This inheritance increases maintainability of code as changes to Entity will automatically update the children classes.

The GameService class does not inherit from any classes and does not have any children who inherit from it.

Each of the Game, Team, Player, and GameService classes have a zero or more relationship amongst each other, which means that they can have any number of instances and to each other.

There can only be one ProgramDriver class, which the SingletonTester uses.

**"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** | **License** – MacOS app required at $20 per server (this is not an enterprise-grade server option) | **Cost** – Free, though hosting or hardware will cost  (Ubuntu is preferred distribution with paid server packages with support)  **Security -** Open-source, segmented OS environment, more secure than Windows. | **Security** - High vulnerability to attack and misuse.  Require firewall.  Cost associated.  Closed-source  **License** – Requires paid Windows server license for each instance ($1609-$6,155 per server for version 2022) if we own the servers. | **Resource**-constrained (CPU, GPU, memory), intermittent connection, not advised for server hosting. |
| **Client Side** | **Browser** – Safari (native), mobile can support other browsers.  **Cost/Time** – must support unique breakpoints per device size and resolution. We will need to pay for more advertising and SEO to be seen on the web versus a hosted app store.  **Hosting** - will need to host all HTTP, CSS, as well as other files and libraries/resource. We will need to pay for the domain which will be determined by the registrar. If we host our own servers, we will need to pay for security management and firewalls, etc. | **Browser** – Lynx (default text-based), Firefox (default for Linux)  **Cost/Time** – Linux has a relatively small community of users when compared to Windows and MacOS. We will need to pay for more advertising and SEO to be seen on the web versus a hosted app store.  **Hosting** - will need to host all HTTP, CSS, as well as other files and libraries/resource. We will need to pay for the domain which will be determined by the registrar. If we host our own servers, we will need to pay for security management and firewalls, etc. | **Browser** – Chrome (most popular), Edge possible.  **Cost/Time** –We will need to pay for more advertising and SEO to be seen on the web versus a hosted app store. | **Browser** – Chrome (Android), Safari (iPhone), Will depend on countries serviced  **Resources** - constrained (CPU, GPU, memory < 1GB), intermittent connection, variable OS type and age, variable hardware. Power-conscious. Small screen size, lack of I/O generally. Frame rate, color gamut, image/sound/size resource limitations, limited physics simulations  **Cost/Time** – UI/UX will need to be reworked for the small screen and inputs from mobile users; this could be a substantial rewrite. We will need to pay for more advertising and SEO to be seen on the web versus a hosted app store.  **Hosting** - will need to host all HTTP, CSS, as well as other files and libraries/resource. We will need to pay for the domain which will be determined by the registrar. If we host our own servers, we will need to pay for security management and firewalls, etc. |
| **Development Tools** | **Languages** – native development would require use of Swift and xCode build tools. Because we are launching a browser-based game, we will be using the APIs available by the Safari browser.  **Cost/Time** – Must pay to be an Apple developer ($299 enterprise license annually)  **Development** – HTML 5 likely one of the languages we would use (works on any modern browser). Javascript is also a likely interactive element in some framework. Hybrid development platforms could streamline distribution to all platforms but carry a fee. Native development not required for web, but decisions about languages such as Java, different calls to native hardware require specific coding per platform  Separate front and backend teams will likely be required for development as these are generally very specific development efforts.  **IDE** – Will not be constrained to a specific IDE as we will be coding to host on our servers. Some IDEs may charge fees, but Visual Studio Code is free for commercial use. | **Development** – HTML 5 likely one of the languages we would use (works on any modern browser). Javascript is also a likely interactive element in some framework. These would not work on Lynx.  Separate front and backend teams will likely be required for development as these are generally very specific development efforts.  **IDE** – Will not be constrained to a specific IDE as we will be coding to host on our servers, however there are a limited number of tools available to develop on Linux systems. | **Development** – HTML 5 likely one of the languages we would use (works on any modern browser). Javascript is also a likely interactive element in some framework.  Separate front and backend teams will likely be required for development as these are generally very specific development efforts.  **IDE** – Will not be constrained to a specific IDE as we will be coding to host on our servers. Some IDEs may charge fees, but Visual Studio Code is free for commercial use. | If we were to develop a native application instead of a web-based experience -  **Android** – Largest market share internationally, kotlin is native language for apps  **iOs** – xCode build tools, Swift, Kotlin, language,  App stores for both Android and Apple (iOs) have differing requirements. Apple and Android (Google) must review all entries.  **Development** – Hybrid development platforms could streamline distribution to all platforms but carry a fee. Native development not required for web, but decisions about languages such as Java, javascript, etc. would need to be made. HTML 5 is a likely choice as well as Javascript for interaction. Different calls to native hardware (such as GPS or cameras, etc) require specific coding per platform  Separate front and backend teams will likely be required for development as these are generally very specific development efforts.  **IDE** – Will not be constrained to a specific IDE as we will be coding to host on our servers. Some IDEs may charge fees, but Visual Studio Code is free for commercial use. |

## 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 that we host the game in a Linux environment as it is free from licensing fee and is considered to be more secure than Windows. We will pay a monthly hosting fee based on usage in a serverless environment, but will have the ability to adjust resources and will have the added benefit of security infrastructure through the hosting platform (AWS, Azure, etc.) I recommend developing the application in a hybrid development environment if we intend to expand to other systems in the future (beyond web-browser clients).
2. **Operating Systems Architectures**:   
   The serverless implementation of Draw it or Lose it will consist of several services in a distributed system which will work together through Application Programming Interfaces to store and retrieve the required files, manage secure details of players and payment methods, and track scoring, customization details, and perform matchmaking amongst users.
3. **Storage Management**:   
   In our distributed system for Draw it or Lost it, we will be relying on a Network Area Storage solution to ensure that multiple services can access storage as our system grows and shrinks to meet demand and future developments. Locally, on the servers, applications will cache common requests, but will make calls to the databases to get required information not in the cache. The cache will have a Time To Live, which will ensure that the locally stored information is refreshed periodically.
4. **Memory Management**:   
   The Linux Server operating platform uses demand paging as the primary method of memory management on the server. When the OS goes to perform a query or operation, it loads only the information that is required into virtual pages in memory. If a request is made for memory that is not in the virtual page, then the processor will throw a fault error to the OS. If the call to memory was invalid, the OS will stop the process from running as it has made an illegal attempt to call memory. If the call to memory was valid, the OS will load the required page into memory. Linux will load partial images of executables into memory in a chain, loading other pages as required to execute commands.
5. **Distributed Systems and Networks**:   
   We will use the distributed system to interface with many different consumer devices, with the ability to expand our client-base in future. The use of web services and restful APIs will allow us to decouple the operations performed in each part of the architecture/system so that we can constantly upgrade each piece without impacting the functions of the other components that service interfaces with. We will also use load balancers and schedulers to launch redundant servers as needed to ensure that outages in different regions have minimal impact on consumer experiences. There may be small increases in latency as servers in different areas fill in for downed servers, but the outages will be infrequent, and Draw it or Lost it will continue to run, which is great. With this architecture, client devices will make calls to our web services and database services to retrieve the required information, authenticate, and store results of games and other changes. We will cache the most used content locally on the user’s device to reduce the bandwidth required, improving the experience and cost for mobile consumers.  
     
   The client devices will run local processing and storage on consumer devices, calling the distributed network endpoints to get or report required data.
6. **Security**:   
   The base security of the bare metal servers in our architecture will be maintained by the cloud providers, but the application security will be managed through certificates and tokens, which will use asymmetric cryptography to authenticate and transmit sensitive data. The service APIs will use the tokens to validate the users and to ensure that the calls to the APIs are authentic. These protections will do a great deal to minimize denial of service attacks, though they will not prevent them entirely. We will, however setup firewalls between the public internet and our services, so that we can whitelist users, and may even setup cleansing services which will take in calls, check against approved (whitelisted) and disallowed (blacklisted) submissions. We can also use these safety measures to prevent repeated calls from Ips, furthering our ability to screen DDOS attacks. We will also hash and salt all credentials and sensitive information, so that the security of our users is maintained.