

Draw It or Lose It

# **CS 230 Project One Software Design**

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 | 3/16/2022 | Jonathan Kleven | Initial Release identifying Executive Summary, Design Constraints, and the Domain Model |
| 2.0 | 3/31/2022 | Jonathan Kleven | Examined and described Server and Client-side criteria and concerns and provided information on development tools. |
| 3.0 | 4/11/2022 | Jonathan Kleven | Detailed Recommendations section |

**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, is looking to develop a web-based game for its already existing "Draw It or Lose It" Game on the Android operating system. Creating a web-based version will help the client serve multiple platforms via a web interface. The game "Draw It or Lose It" has an extensive library of images that, for each Game or round, will render images from that library to serve as clues to the puzzle for each team or player to guess in the time allowed.

## [Design Constraints](#_2et92p0)

The client already has a version of this game developed for the Android operating system.

**Technical Constraints**

The client has identified aspects of the game that will need to be considered for developing the Game in a web-based version.

* Only one game instance can exist in memory at any one time.
  + A Game consists of four rounds of one minute per round.
  + If a Team has not successfully guessed the puzzle at the end of the round, the available Teams are given 15 seconds to solve the puzzle.
* A Game must have the ability to have one or more Teams involved.
* Unique Game and Team names allow users to check if a name is in use when creating a Game or Team.
* Multiple players can be assigned to a single Team.
* The Game will render images from a library of stock images to offer to the Teams as clues to the puzzle.
  + Images will render at a steady rate and complete rendering at 30 seconds.

The primary language that will be employed will be JAVA. There are many similarities between JAVA and applications written for Android operating systems; namely, the languages used for Android development are Kotlin and JAVA.

Moving "Draw It or Lose It" from an Android application to a web-based version will require hosting the Game for use on the internet. It needs to be able to handle the Game and its requirements and the traffic that will be making requests to execute the play of the Game. It will also need to be able to house the client's library of stock images and its data structure. The image size and resolution of the stock images should also be considered.

Another item of consideration is if the web-based Game and the existing Android application will be able to communicate/integrate, i.e., an Android team member playing and interacting with other or opposing teams using a web browser on Windows or Mac operating systems.

**Business Constraints**

Assuming the Android app allows its users to connect and play over the internet also considers that the client may be familiar with hosting requirements. Additionally, based on the development of the web-based version, the hosting requirements will need to be modified to handle both Android-only users and those accessing the Game via a web browser. When this is considered, the hardware required modifications for the web-based Game will need to be completed near the end of development for testing purposes. Finally, the client can choose to host the web-based Game separately from the Android users or host them together. Hosting the Game for Android users and new web-based users will align with the client's goal of serving multiple platforms.

## [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)

Figure 1, shown below, is the Unified Modeling Language (UML) of the game "Draw It or Lose It" structure. There are several classes to identify and how they interact with one another.

The main program will be run via the ProgramDriver class. This class will execute the Game and allow users to create games, teams, and players. In addition, it will employ the use of the SingletonTester class to ensure any inputs to the Game, team, or players are unique and only used once.

The Entity class is an abstract class, meaning that it cannot be instantiated. However, methods of the Entity class can be inherited and passed down to its child or subclasses. We have subclasses Game, Team, and Player classes within the Entity class. Several methods inherited by the subclasses adhere to the Don't Repeat Yourself (DRY) guidance. Such as getId() and getName().

The Game class, a subclass of Entity, will hold all the information about a game. First, this class will define a game as having a unique ID and name. Next, it lists *active team*s referenced when a new team is added with the addTeam method. The addTeam method will first read the list *teams* to ensure another team of that name has not been used. It also includes an instantiation of the Game class referenced in the GameService Class discussed further. The final method is toString which, when called, overrides the Entity parent class toString and will display a game's name and a unique id.

The Team class, a subclass of Entity, will hold the information about a team. It begins with a list of players that contains each player's name on that team. This class will define a team as having a unique ID and name. The addPlayer method will first check the list of playersto ensure another player of that name has not been used and, if not, will add a new player to the team list. The final method is toString which, when called, overrides the Entity parent class toString and will display the team's name and a unique id.

The Player Class, a subclass of Entity, will hold the information about a player. A player will be defined in this class as having a unique ID and name. The final method of the Player class is toString which, when called, overrides the Entity parent class toString, and will display a player's name and a unique id.

The GameService class holds the information regarding games. Like the previous classes, it has a list of active games. It provides a starting point for the unique IDs given to games, teams, and players. It includes a private constructor that only permits the current class to instantiate it, though it is externally accessible. Giving the option to be externally accessible allows only one private constructor to be called only once. Once a game is created, it is added to the list, and any additional games will be checked against the list *games* to ensure each game id and name are unique. We can then get game information via the unique ID or name. Finally, it includes the method for teams and players id's to be incremented every time a new team or player is added.

**"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.**

**Figure 1: Unified Modeling Language of "Draw It or Lose It"**

## [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** | The MAC operating system offers a range of out-of-the-box features desirable to many users. It makes up almost 10% of the computing market. The MAC operating system does have a proprietary application to manage server development called MAC OS X Server. This software does have an additional fee to purchase. However, this OS offers a consistent interface and routine updates. Generally speaking, Malware is less of an issue with these systems. However, hardware upgrades are complex if at all possible. As demand increases and the hardware is not prepared to meet those demands, a new computer with the specifications needed and configuring it with the software and source code for the server will be required before the upgrade can be finalized. | The Linux operating system can be a bare-bones operating system, which means that it is not bound to the restrictions of large corporations like Microsoft or Apple. However, that too is a weakness as much of the server development will be from the ground up. It provides the freedom to configure however you wish to meet the client's needs. There is a small use case for business and a steep learning curve for configuration. If cost savings is a goal, then this operating system has a large community of support and open source solutions that, with sufficient knowledge of how Linux works, can meet the same standards used by Apple and Microsoft | The Windows operating system makes up 90% of the computing market. This makes security and protection more prone; additional software and associated fees are needed to guard against malicious threats. Despite that, Windows offers many corporate-level connectivity options out of the box. Unfortunately, this only comes with the Pro version of the operating system, which is an additional cost. | Mobile devices have come a long way in their development since becoming mainstream. They are more pocket computers than telephones. The challenge with mobile devices will be to assess the hardware requirements needed for hosting. It may be possible to host a server, though its hardware limits it, and future development will only go so far. |
| **Client Side** | The MAC operating system will have substantial support to provide the client-side of development. Most commercial-level software will be compatible. Most development languages are supported, including Android Studio, an IDE to support the development of applications for Android operating systems. The added benefit of user interface consistency will ensure those familiar with MAC a shorter learning curve for its use. MAC also has access to integrate the most common browsers used seamlessly. | Much of the Linux operating system software is open source and available in many of its configurations. However, the steep learning curve of applying Linux-based solutions cannot be ignored. Linux is compatible with many IDEs. However, testing requires additional configuration, such as emulation. However, there may be a disconnect between the emulation software and the actual software, leading to testing and product rollout complications. Linux has its own web browser and is compatible with some browsers like Google Chrome and Firefox. | Development on Windows will have similar or better support than the MAC operating system. In recent years, its proprietary web browsers have left much to be desired from users. They tend to seek out the more common browsers, with which Windows is seamlessly compatible with all of them. | Many of the browser applications found in Windows and MAC can be used on a mobile device, though with limited functionality due to the hardware and size requirements of the mobile device. Additionally, many open source and paid applications allow for client-side development, though again are limited by the hardware available to the respective device. |
| **Development Tools** | The primary proprietary software available is SWIFT, boasting easy-to-understand and read language with many code checking functions. In addition, most IDEs and tools are available for MAC and compatible such as Python, Java, HTML, CSS, Javascript, and C. | Linux is compatible with many open-source variations of standard PC/MAC IDEs. In addition, it can handle languages such as Python, Java, HTML, CSS, Javascript, and C. | Similar to MAC, Windows is compatible with the most popular IDEs available. | Development tools are available for use with mobile devices. However, they will be limited by the device's OS and hardware. Often, emulation is the method used to create the application, which may not accurately portray the IDE or language its emulating. |

## 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**:

We recommend that the operating platform that will fill the clients' greatest needs while not compromising the cost of getting it up and running would be a Windows Operating system. The Windows operating system is used by the largest demographic of computer users and supports development in any language and for any function. In addition, the hardware available for Windows is customizable to suit the game Draw It or Lose It's development.

1. **Operating Systems Architectures**:

For the initial development, we recommend that the system architecture be a Windows Pro 64-bit operating system machine using the current version of an intel i-powered processor with a minimum of 8GB of Random Access Memory. This will give the developers the necessary flexibility to develop the application while not sacrificing hardware performance.

At the onset of testing and application rollout of the application Draw It or Lose It, we recommend the client utilize the services of serverless cloud computing. Since the client already has some experience and success with its android application, opening the game up to all platforms regardless of the operating system is an enormous task. One that cloud computing and serverless architecture are well suited to handle. Once development has reached its test phase, it will be essential to experiment with each aspect of the game and how the user will view it on their respective platform. Handing off the server responsibility to a cloud provider offers the client more freedom to work on the game rather than the game's communication. The added benefit of a serverless architecture is that any operating system for the server can be used, with maintenance, testing, and troubleshooting being handled by the third-party cloud service. This has the benefit of the cloud provider using a system that they know well and optimizing that system to meet the client's needs better. In addition, this recommendation allows for smooth scalability. If the game application becomes increasingly popular, the provider will be able to handle the additional request. While this increase will come at an additional cost to the client, this scalability feature ensures that the pay scale is adequate to the demand. This way, the client is only paying for what they need, allowing them to focus on user experience and game performance and features.

1. **Storage Management**:

During application development, the application data storage will be local to establish proper communication between the server and the client. We will need a minimum of 2 Gb of storage space to ensure enough room for images and the application data library. This is subject to change based on the needs of the application.

Once initial development is complete, testing begins moving to serverless cloud computing. Hosting by a third party such as AWS (Amazon), Google, or Azure (Microsoft) will better handle the backend storage needs. The most significant benefit to this system is only paying for what you need. Scalability is crucial to meet the increased demands of users. The options and pay scales will adequately reflect that change. With the added benefit, the third parties take on the responsibility of storage needs, maintenance, and troubleshooting.

1. **Memory Management**:

To control the management of memory and deliver consistent performance to the user, we must first establish the client side of the needed images and game data. This can be in the form of image sizing. Each of the photos in the library needs to be of similar resolution to ensure that the images are transferred to the user while not eating up the performance of inflated or oversized images. Once the images sizes and resolutions are consistent, we can then discuss the execution of image data to the user with content delivery networks (CDN). This may be a function that the serverless hosting provider offers or an additional service. A CDN provides a cached version of the image data at many geographical locations to ensure the data is physically closer to the user that is requesting it. This increases page load times and reduces latency. In effect allows the user to experience a more seamless game experience without waiting for the code and data to finish executing. In addition, it gives the user a faster display and load times of the game information and proper rendering image during their time allotment.

1. **Distributed Systems and Networks**:

Suppose the client decides to move forward with our recommendation of using a serverless cloud model to support production of the game, Draw It or Lose It. In that case, we will be able to provide access or communication to the various platforms and operating systems using distributed systems and networks. In a basic sense, distributed systems are computing assets not physically located within the company's property. Utilizing this network will better serve the global users who engage with the game. This network will appear to the user as one cohesive application because they will be able to effectively communicate and converge all user inputs and requests in one action. As we stated earlier, the benefits of a serverless architecture are that the maintenance and troubleshooting are the responsibility of the serverless cloud provider. However, another advantage is that these providers also have a global network that will reduce the latency or time of the user input and requests with the application servers.

For example, consider that if the client and server were housed within one location, there would be a noticeable time difference in application load times for a user in the same state as the server versus a user located on the other side of the globe. The serverless cloud provider would reduce this latency of application load time by utilizing distributed systems and networks with redundancy servers acting as intermediaries to users who have a greater distance to the application server. Essentially, ensuring the heavy or long-distance work is carried to the user with excellent efficiency.

1. **Security**:

Security of the user's information should always be at the forefront of our game application development. To ensure that the users are protected, we must think of the data we will be collecting and the methods we will employ both within the application and our serverless distributed system. Let us not forget that making sound security decisions within our game application is not just for the users; it also protects the company's asset, Draw It or Lose It, and enables it to continue to perform its functions unencumbered.

To ensure a positive user experience, we must inform our users of methods we wish to use regarding their user logins and passwords. For example, a two-factor authorization method is a popular control to verify the users and their login. First, this method asks for the user's login credentials. Then, it sends a confirmation request via a medium of the users choosing, ie text or email, to which the user will respond to close the loop of authentication. Additionally, we can request that our users create unique passwords that are more complicated for a malicious person or machine to guess. This information will be provided to the user upon registering for the game. Additionally, suppose we recommend a specific time for the user to update or change their password. In that case, existing users can be notified of this change upon logging in, where they would be asked to change their password.

To protect the client's assets, we will need to consider security at each stage of development and identify where sensitive information should be stored. While we could rely in some cases on the serverless cloud provider for this protection, it will make two companies vulnerable should there be any lapse or unknown exploit used by uninformed users or malicious persons. Therefore, we should consider methods that will protect the user, the client, and the serverless cloud provider, such as data encryption and hashing.

Encryption is a security method that masks data communication by encoding it. Unless the data is decoded with the proper decryption method, the information will be viewed by anyone not authorized to view it as a mash-up of numbers, letters, and symbols with no rhyme or reason to information or structure. Information will be encrypted, sent, received, decrypted, and processed appropriately.

In addition to encryption, we recommend user data be hashed on the server-side. Hashing of user data is a method that avoids storing plaintext data such as User: John Doe, Password: JohnDoe123. Hashing is a one-way process of converting the user data's plaintext to a fixed-length string of characters. For example, previous user information will be seen as 9b9s6x7c8vxc9vxc7453zx2xc7v. The user would enter their credentials upon logging into the game. Plaintext information is converted to a hash before any requests are sent from the user on the client side to the server. Then the hash is validated with the information on the server to confirm the user's login. If the encryption of the data is made vulnerable, the only information that will be readable will be the hash value. The hash value cannot be used to determine the user's plaintext equivalent.

Using encryption and user data hashing techniques during the game application development will increase the protection of the users and the game. Couple this with the security guaranteed by the serverless cloud provider, and we will not only be providing a safe and secure data system. Still, we will ensure that the integrity of the game application will be more complicated to trace and make vulnerable.