

Draw It of Lose It!

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[CS 230 Project Software Design Template](#bookmark) 1

[Table of Contents](#bookmark1)**2**

[Document Revision History](#bookmark2)**2**

[Executive Summary](#bookmark3)**3**

[Requirements](#bookmark4)**3**

[Design Constraints](#bookmark5)**3**

[System Architecture View](#bookmark6)**3**

[Domain Model](#bookmark7)**3**

[Evaluation](#bookmark8)**4**

[Recommendations](#bookmark9)**5**

## [Document Revision History](#bookmark10)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/13/24 | Clifton Brann | Initial Changes and data entry. |
| 2.0 | 07/31/2024 | Clifton Brann | Edited Development requirements . |
| 3.0 | 8/13/2024 | Clifton Brann | Edited recommendations |

## [Executive Summary](#bookmark11)

The Gaming Room has come to CTS to take their Android based game app Draw It or Lose It and make it a web based multi platform game. Each game consists of 4 one-minute rounds of play, where the computer renders a picture of the puzzle clue over the course of 30 seconds from a large library of stock images. If the one team does not get the puzzle answer before the one minute time limits runs out, each of the remaining teams will get 15 seconds each to guess what the puzzle clue is and win the points for the round.

## Requirements

The primary problem is that the Gaming Room doesn’t know how to adapt the current version to a web based application, which is where we come in. The other requirements given by the Gaming Room are: the ability to have multiple teams able to join each game, have multiple players on each team, each game and team name must be unique and be able to be checked for uniqueness when choosing a name, and only being able to have one instance of the game running at a time.

For game play, there must be 4 one minute rounds, with followup 15 second rounds for the other teams, if the first team whose turn it is fails to guess the clue correctly. The images have to be rendered or revealed over the course of 30 seconds, and the clue and the image have to correspond and make sense. The images have to be pulled from a large library of stock images, and be catalogued to correspond with clues that are a phrase, a title, or a thing.

## [Design Constraints](#bookmark12)

In the redesign of this software, the game should be developed to be able to interact with Windows, Mac OS, and the Web.

The multiple players and teams playing in the same game must allow for players and teams from the multiple platforms to play together.

Since player and team names need to be unique, there needs to be a way to tell teams and players if the name they want to use is already in use.

There also needs to be a unique ID generated for each game, so that every game has a unique identifier, and that way the end user can only play one game at a time.

The image library also needs to be built, indexed, and updated regularly to keep the game fresh.

## [System Architecture View](#bookmark13)

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](#bookmark14)

In the following UML diagram, you will see that the Entity class has the most arrows going to it, this means that it is a parent class, and that the three classes that point to it are the child classes of it. This means that all the information in the Entity class is inherited by the the three child classes, Game, Team, and Player. This will allow the game to be more streamlined and compact without having redundant information and code repetition.

The diagram also shows that Game, Team, and Player all being connected, this means that they will share data as needed from Game to Team, and from Team to Player, but not the reverse. GameService sends data to Games, and gets all the IDs it needs from the other classes.

The Program Driver class is by itself with SingletonTester, so it can use the SingletonTester class to assure that only one instance of the game is running at any given time. The sole purpose of the SingletonTester class is the make sure that player IDs, game IDs, and team IDs are all unique and non repeated.

**"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](#bookmark15)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Advantages:  Mac OS has hit built in security, due to it being a closed source OS. There is a lot of resources and support for Mac hardware ad software. Servers are easily accessible and configurable for multi platform support.  Disadvantages:  Mac is the most expensive to set up in terms of hardware, and the hardware selection is limited. | Advantages:  Linux is an open OS, which is well supported with reference resources. It has well integrated comet. Administration functions. This platform also supports a cooperative work environment.  Disadvantages:  Linux can be complex to operate efficiently. Sometimes updates can have unexpected results. This platform does nat have a GUI, only runs trough a command line prompt. | Advantages:  Windows os one of the most user friendly OS, and it supports a lot of applications and connects to almost everything. It also has a lot of support from third party apps. Because it is established there is guaranteed long term support.  Disadvantages:  Windows has a high opportunity for user error to cause problems. It is also very vulnerable to malware, spyware, and viruses. Hardware for servers is readily available and relatively inexpensive. | Advantages:  Mobile devices are very portable. The program code should be very secure as it is not accessible on a mobile device. Usage statistics are easy to collect from mobile devices.  Disadvantages:  The different operating systems on mobile devices can cause variations in program functionality. The security on mobile devices can also cause problems with overall functionality. |
| **Client Side** | On the client side most common browsers and tools are supported and usable on Mac OS. However the hardware required for servers is some of the most expensive available. | Linux has the lowest cost to purchase equipment, but will will require the most time and expertise to set up and run. | Windows is probably the most intuitive system to setup and operate, so it is more cost effective to operate then Mac, but more expensive than Linux. | Because mobile devices are setup to receive updates, updating the app will be easier than other systems. However, due to different OS languages, the app would have to be written and maintained in the separate languages, |
| **Development Tools** | * Programming language would be Swift * IDE would be Xcode, VisualStudio Code and Fierce Code. * Other languages supported by MacOs are Python, Ruby, Java (to name. only a few) | * Programing Language would be C * IDE would be Eclipse * Other supported languages are Java, Python, and JavaScript (to name a few) | * Apparently C# would be a primary programing language * The IDE would Be Visual Studio * Other languages supported are: Java, Python, and many others | * Java would be a primary programing language |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Tools** | From the development side, the programming language will be Swift. Commonly used IDEs would be Xcode, VisualStudio or Fierce Code. Some of the other languages supported by MacOS would be Python, Ruby, and Java. | Most Linux programming is accomplished in the C language. A common IDE for Linux is Eclipse. Some of the other supported languages are Java, Python, and JavaScript. | The programming language for Windows is C#. A good IDE for this would be Visual Studio. Some of the other supported languages are Java, Python, and many others. | Since. Java is mobile friendly, it would probably be the easiest language to use for programming the servers. Android Studio or Xcode would be good IDEs for this sort of situation. Some of the other supported languages are Python, Kotlin, and Objective-C. |

## 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**: A Windows based operating platform or server will be the best choice in terms of cost, flexibility, and setup. Since Windows is well established as a hosting system, and the equipment is both versatile and readily available, the technical requirements, and associated costs should be reasonable when compared with most other systems hardware and ease of use.
2. **Operating Systems Architectures**: The event driven mediator architectural pattern would be the most efficient for this game. The game will require multiple events to be coordinated in order to make the game function properly. For example a user has to register their user name, join a team (potentially having to register that name), then find (or be assigned) another team to play against, then the image the teams will see needs to be chosen, downloaded to the various mobile devices and rendered over 30 seconds. Then the scoring needs to be done, and the middle part of the process is repeated for each round of play, until the conclusion of the game, where the winning team is announced and points assigned to all parties who played.
3. **Storage Management**: A multipart storage management approach will be best. Storing the game operating files on the various mobile devices, desktops, and laptops will make the game run most efficiently. However hosting the image files centrally on the network server would simplify the task of updating the images, and minimize the impact of the data stored on the various devices. Storage is more of a problem for the mobile devices, as they tend to have more limited storage space than either desktop or laptop devices.
4. **Memory Management**: Since it is important to optimize the code to be as complete and compact as possible to minimize the impact on the users deviceThis concept is is more important for mobile devices, and to a degree laptops, have both limited processing and battery life. It will be very important to minimize the amount of code that is run at any given time, to keep the volatile memory more available on the device, to be able to render the image properly in the correct amount of time. Resizing the images to a standard size, and compressing the data so that the downloaded file size could be minimized would be a key consideration in managing the memory for this game. It would also be important to utilize lazy loading. This is a concept which loads content, such as images at it is needed, instead of all at once. For example, if a webpage has twenty four images in a list format, but only the first four a visible on a mobile device, only the first four images would be loaded when the webpage as loaded. The remaining twenty images would be loaded in order as the user scrolled the page down, and the image was “revealed” on the screen as the user scrolled to it.
5. **Distributed Systems and Networks**: It makes sense to use a central sever system to host the game files, for ease of maintenance and updating of image files. However, it may be more cost effective to utilize cloud services to both host the game files and image files, which charge by the number of gigabytes stored in the cloud drives and terabytes transmitted across the internet. It would also be worth while to use a CDN or content delivery network to transmit the images more quickly then they could be served from a central server unit. A CDN would have servers with cached images scattered across the country or world and decrease the access and download time though both proximity to the users and through specialization of sending large files.
6. **Security**: Since the application will be run on Windows based equipment, it will be vital to run the appropriate hardware and software firewalls and proper antivirus protections for the equipment. One of the drawbacks to software that can interface with almost everything is that there are a lot of ways for it to become compromised, so a lot of thought needs to be given to how best to keep the hardware and software systems safe. It would be important to build in protections to the code and server systems, by use of the principle of least privilege. This means that end users would have the least privileges to access anything on the servers, in the game code, or any other system connected with the game hardware. System administers and developers would have the most privileges to access sensitive parts of the system and code, because it is their responsibility to maintain, upgrade, and patch the software as needed. At the same time it may be very difficult to defend the system from all outside threats. By writing code that is resistant to a stack overflow attack is a good way to maintain system security. At the same time there is very little defense against a DDNS, distributed denial of service attack, if the attacker is very intent on choking off the access to the game servers. Another facet of security is the human factor, the people on the operations side of the software and hardware have to savvy enough to know not to allow phishing or other social engineering attacks on their data, logins and passwords. Unfortunately it looks as though security is a cat and mouse game, to a certain degree. Just when it seems like the system is “totally secure” and proof against any attack, someone will find a way to exploit some software hole, security flaw, or invent a new way to intrude on a system for whatever purposes they have in mind.

**References**

IONOS Digital Guide. (2019, August 12). *Linux vs Windows – the big server check*. IONOS Digitalguide.

<https://www.ionos.com/digitalguide/server/know-how/linux-vs-windows-the-big-server-check/>

‌ keycdn. (n.d.). *CDN Storage*. KeyCDN

<https://www.keycdn.com/cdn-storage>

Lavieri, D. E. (2019). Hands-On Design Patterns with Java. Packt Publishing.

<https://mbsdirect.vitalsource.com/books/9781789809954>

Sacca, Suzanne. “A Guide to Optimizing Images for Mobile.” *Smashing Magazine*, 22 Oct. 2019,

<https://www.smashingmagazine.com/2019/10/imagekit-guide-optimizing-images-mobile/> Accessed 13 Aug. 2024.

*What is Lazy Loading | Lazy vs. Eager Loading | Imperva*. (n.d.). Learning Center.

<https://www.imperva.com/learn/performance/lazy-loading/>