

Draw It of Lose It

# **CS 230 Project Software Design Document**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Document**](#_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 | 05/23/21 | Carmen Kingery | * Updated the Executive Summary * Updated the Design Constraints * Described UML class diagram |
| 2.0 | 06/02/21 | Carmen Kingery | * Updated the Evaluation table |
| 3.0 | 06/20/2021 | Carmen Kingery | * Updated the Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

Our client is The Gaming Room. They have a new game out, called Draw It or Lose It, that is currently only available as an Android app. They are wanting to create a web-based version so that it can be available to a multitude of platforms. While being able to serve multiple platforms, there can only be one instance of the game at a given time. Within the game, the application should be able to support any number of teams, and each team will have multiple players. And lastly, game and team names must be unique within the current instance of the game. The Gaming Room is unsure of how to set up this environment. Therefore, they have reached out to us as their consultant.

Requirements:

* 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.
* Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

* Language – we need to pick a language that is versatile so that our game will be able to run on a variety of different platforms.
* Database/storage – we need to keep storage space in mind. We are making a simple web-based game, not an entire tier 1 video game with endless storage.
* Timescale – we only have a week to accomplish planning, developing, *and* testing this project.

Proposed solution:

We are going to be using Java as it is extremely versatile in the different platforms that can run it. It also has some great built-in libraries for game design and is easy to modify with updates later down the road. As far as the storage goes, we will be using in-game storage since nothing needs to be saved from one game instance to another. Because of the short time span we have been allotted for this project, ideas need to be streamlined and work prioritized in an order of maximum efficiency. Also, to accomplish the single instance of the game, we will use the singleton pattern to make sure no other instances are being created while one is already running. We will also separate the game, teams, and players into their own classes. The game class will hold a list of the teams signed in and the team class will hold a list of the number of players signed into it, to make sign in and team divisions easy for the players.

## [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)

In the UML diagram below, we see a break down of how this program will be organized. We have a parent class, named Entity, that holds an id and a name. The children classes, Game, Team, and Player, will all inherit these attributes. This is to ensure each game, team and player all have unique names and identities. On top of that, it helps reduce redundancies in the code, keeping the program readable. Each of the children classes, as well as the GameService class, are closely associated with each other because they each use one or more methods from the others (e.g. the Team creates Players, a Game creates Teams, and the GameService creates Games). The other classes can use these methods, however, they are not able to see what really goes on in the method once they have called it. This demonstrates abstraction and is once again used to keep the code base clean. We also have a SingletonTester class that ensures we only have a single instance of the GameService active at any one time. The ProgramDriver is where the game options will be displayed and how all the other classes will be accessed. The ProgramDriver goes through the SingletonTester to create a single instance of the GameService which will in turn create games with teams with players, who will all call on the Entity parent class to create their individual attributes.

****

## 

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Does offer a few servers for hosting a website but they can be difficult to find.  Licensing costs more due to not having very many servers | Offers multiple different server-based plans for hosting your website.  The server licensing costs are extremely cheap ($2-$5/month). | Also offers multiple different server-based plans for hosting your website.  Licensing costs more ($8-$10/month) | There are many cloud server options for mobile devices.  Costs vary widely (anywhere from $5/month to $650/month) |
| **Client Side** | OS is very user friendly.  No expertise needed.  Updates are automated.  Requires Apple-only hardware which is pricy. | OS is free.  Very  customizable.  Steep learning curve.  Very versatile in what platforms it can be run on. | OS is extremely open and flexible.    Easy to learn.  Updates are automated.  Has minimal hardware requirements.  Is also very versatile with its platforms. | Do-able but not ideal for development.  Is more stringent in what platforms it can run on.  Will take extra time to make compatible with the desktop website due to it being a completely different machine. |
| **Development Tools** | IDE: XCode as well as the IDEs for Linux.    Tools: EverWeb, Seashore, Coda, Nvu… all very reasonable in price.  Languages: Objective-C, Swift, PHP, C Language, HTML. | IDE: Eclipse, Atom, Code::Blocks, PyCharm, NetBeans…  Tools: PhpStorm, Visual Studio Code, Sublime Text, Vim… also reasonable in prices.  Languages: C++, Python, Java. | IDE: Notepad++ as well as the IDEs for Linux.  Tools: Sencha Ext JS, CodeKit, TypeScript… all reasonable in price.  Languages: C#, C++, C Language, JavaScript. | IDE: Visual Studio is currently the best for this.  Tools: BootStrap, DevTools, Blisk, Lighthouse… also reasonably priced.  Language: C++, Ruby, Java. |

## Recommendations

**Operating Platform**: For the purposes of expanding Draw It of Lose It to be played on a multitude of different platforms, I would recommend using a Windows server. Not only is Windows extremely versatile with its platforms and usage, but it is also very affordable. On top of that, Windows is widely used and understood by developers, which means there will be little to no learning curve for your team when working with the server.

**Operating Systems Architectures**: Windows servers use the client-server architecture. That is, the server, connected to the database and all the resources, waits until the client asks for something and then responds. It just sits there like a cashier at a coffee shop, waiting for a customer to ask about the coffee they offer or to place an order. In this fashion, our application operates on minimum bandwidth and maximum operating efficiency.

**Storage Management**: For our purposes, all we need to store is our simple game application as well as the file of high-definition images. These will be stored on the server as long-term storage. I would recommend everything else that happens within the game uses in-game memory/storage because nothing technically needs to be saved from one game to another.

**Memory Management**: The approach I would take to manage the game’s memory efficiently would be to have the application select and load the first five pictures into a waiting queue at the creation of a new game, ready to be used. Then as the game plays out, the first photo is pulled from the front of the queue and sent to the screen while a new photo is pulled from deep storage and placed at the end of the queue. The used photo is then discarded as the next photo is presented. This cycling through of the photos would continue till the end of the game is reached, optimizing our memory usage while keeping the game running at its proper, preconceived pace.

**Distributed Systems and Networks**: Because we are using a server as our game’s running platform, communication can be achieved with any device that is capable of accessing our server. And because our server will be a Windows server, that pool of connectable devices is quite large. Connected platforms will be able to communicate with each other, regardless of what platform they are using, on the server. They cannot communicate directly, instead the server communicates with each individually as a middleman. Because of this setup, connection with one of the members could be lost if they were to lose power or have a bad network connection. However, the game would continue to play for the remaining players.

**Security**: We will be utilizing a username and password system so that if a player loses connection for whatever reason, they will have a way of reconnecting so long as their game is still going on. These usernames/passwords are discarded upon termination of the game and cannot be used for logging-in while a game is not in session to protect against malicious intents. In this way, player accounts will created for the duration of the game and deleted at the conclusion of the game. Windows also includes its own set of firewalls to protect against viruses and the like.