

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**Domain Model 3**](#_Toc115077324)

**Evaluation 5**

**Recommendations 8**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/28/2025 | Michael Foster | Created executive summary, requirements, design constraints, domain model, evaluation and recommendations. |
| 1.1 | 04/15/2025 | Michael Foster | Updated Evaluation. |
| 1.2 | 04/17/2025 | Michael Foster | Updated Recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

The Game Room has an application Draw It or Lose It currently available on Android. They are looking to move the application to a web-based service to provide cross platform capabilities. The intent is to maintain the things that made the initial application successful and expand it to additional markets.

## Requirements

* The game must be developed to run in a web-based environment.
* The game must have the ability to have multiple teams.
* The teams should have players assigned to it.
* The game and team names should be unique.
* Only one instance of the game can exist at a time.

## [Design Constraints](#_2et92p0)

Internal system hierarchy: This application requires that there only be one instance of the game in memory at a time and that those games maintain their internal components. This is possible using the singleton design pattern and fits a web-based application well. The host will maintain a service that provides the ability to create a game. Once a game instance is constructed it will be able to construct teams that are aligned to it in a one-to-many relationship. In turn each team that is created will have the ability to construct instances of players. By having each instance manage the instance under it, control over unique names and management of each component can be maintained easily.

Compatible language: In the initial inception of Draw It or Lose It a language was chosen to meet the requirements of one operating system. Since this application is now being moved to a web-based service the language utilized is going to be important. It is important that the backend logic we are developing will be compatible with any operating system as we do not know what type of webserver it will be deployed on. The most suitable options are C# and Java, but since the team at CTS has not specified that they will be utilizing a .Net web Framework for frontend development in Java will be the best choice.

## [Domain Model](#_8h2ehzxfam4o)

ProgramDriver:

The program Driver holds the Main() and therefore will be the class that drives the program.

SingletonTester:

It is a class that provides unit tests for the singleton design pattern.

GameService:

This class holds an instance of the game and contains fields and methods to ensure that the management of the game. This includes maintaining the current game as well as issuing unique game, team, and payer ids. There will only be one GameService at any given time.

Game:

This class is for an instance of a game. It holds the teams that are playing in the game and has a method to add additional teams. There can many Games that exist in the Game Service, but teams that are stored in this game only exist in this instance of the game.

Team:

This class is for an instance of a team. It holds the players that are on the team and has a method to add additional players. There can be many teams in the game instance that this instance belongs to, but the players that are stored in this instance only belong to this instance.

Player:

This class is for an instance for an individual player. This instance is stored in the Team instance that called its constructor.

Entity:

This class is a super class for Game, Team, and Player. It reduces redundant code by maintaining the id, and name. Additionally, it has methods that act as accessors for those private fields.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac OS X Server is a Unix based operation system that comes with Apache web server software pre-installed so it could host the website. The advantages are that Mac OS X Server is considered user friendly and is known for its security. The disadvantages are that it is largely depreciated and not typically used to host large public facing websites due to high cost, especially when scaling as each hardware device requires a separate license. | Linux is open-source and provides many different OS options depending on the use case of the server. The OS that would be best suited for this is most likely Ubuntu. The advantages are the ability to pick the option that is best tailored to the design as well as significantly lower cost making it easier to scale. The disadvantages are a steeper learning curve as the developers will need to know the operating system that is chosen. | Windows Server is a proprietary OS developed by Microsoft. It comes with built-in features that will allow it to host the website. The advantages of Windows server are the wide range of available tools and optimization for anything developed with the .Net ecosystem. The main disadvantage is going to be cost. Windows server license costs would make it difficult to scale overtime. | Mobile devices cannot be utilized for the server side of this project. Their OS are client focused and are not designed to host multiple user sessions primarily because their hardware is too limited to support these types of operations. |
| **Client Side** | Mac can run multiple web browsers but has one that is not available on other platforms, which is safari. Since the application will be delivered as an HTML interface running inside of the web browser the application will leverage HTML, CSS, and JS which are platform-agnostic. This means that the bulk of the code is identical across platform and depending on testing can be tailored to ensure that the best delivery is achieved in Safari and when run on a Mac OS. This reduces costs but incurs extra testing time to ensure the best delivery. Additional expertise will be needed as Safari is not as common as other browsers and will require some specific knowledge. | Linux supports numerous browsers and does not have a single default browser. Development for Linux is almost identical to Mac and Windows. The front-end development is focused on the browser. After the front end is designed it will need to be tested on different browsers running in Linux. This will allow the developers to identify any features that are not presented correctly and modify JS to render correctly on that operating system/browser. This will require minimal additional expertise or cost. | Windows can run multiple web browsers with Edge being the default. Most of the code will be the same as other desktop-based OS, since the application will be delivered as an HTML interface running inside of the web browser.  HTML, CSS, and JS can render slightly different in different browsers so specific testing in all browser in windows is recommended. This will increase the time required but will deliver a better product. The cost of developing is the same as windows or mac and minimal additional expertise is needed to tailor presentation. | Mobile Devices will require the most time, cost, and expertise. The bulk of the application can be written in one language Java and most of the logic can be reused, but the amount of tailoring for IOS and Android will be significantly more than tailoring for browsers. To maximize code, reusing different frameworks will be utilized for different OS this will require expertise not only in Java but in those specific frameworks for the developer to access platform native features such as touch screen, camera, and loading icons. This will also increase costs as it will most likely require different teams to tackle the different OS. Additionally, there are licensing costs associated with developing on different platforms with Apples being significantly more. |
| **Development Tools** | Languages: Objective-C, Swift, JS, Java, C#  Tools: Xcode, IntelliJ, VS code  Developing applications for Mac requires a Mac as some of the tools such as Xcode are platform specific. Additionally, the tools have different licensing costs and Apple requires team members to work under the Apple Developer Program which has an additional cost. | Languages: Python, Java, C#, C++, JS, PHP  Tools: Eclipse, PyCharm, VS code  Developing applications for Linux is more cost effective as most libraries are free and open source. Some of the tools do require license fees for professional versions. Using open-source tools could result in depreciating dependencies that would then need to be rebuilt/managed internally. | Languages: C#, Java, python, JS  Tools: Visual Studio, VS code, IntelliJ  Developing for sale applications for Windows professional versions and licenses which will raise costs. Additionally, if developing primarily for windows the team should consider developing in a .Net language for improved efficiency. | Languages: Java, Swift, Objective-C, JS, C#  Tools: Android Studio, Xcode, VS code, Visual Studio  Developing applications for mobile devices will vary greatly depending on the intended mobile platform. For this type of development professional licenses for IDE will be required and most likely multiple teams will be needed for specialization for specific platforms, or a language with suitable integration frameworks will be needed. The choice of language and number of platforms will be the primary factor when it comes to cost. |

## Recommendations

1. **Operating Platform**: I would recommend two different platforms be used on the server side to host different parts of Draw It or Lose It. I would first recommend that AWS EC2 instances running Linux, preferably Ubuntu for robust security and large support, be used as your web server. This is a cost-effective platform that will allow for scaling up and down to save costs as demand increases and decreases. For reference files I recommend hosting them on a on-premises Linux server due to low entry cost and low cost maintenance. This option will reduce overall cost to continually run as you should only need to run one server to provide these resources for the web servers.
2. **Operating Systems Architectures**: For this system a client-server model using microservices will be the best approach. The client-server model will reduce redundant code development and improve performance as more system heavy tasks will be performed on the server and rendering will be the biggest lift the client side will be responsible for. Additionally, microservices should be utilized in this design because most of the requirements can be defined as simple tasks that can be developed independently allowing for a more resilient system and easier troubleshooting when errors occur. Using microservices will also open the opportunity to leverage serverless functions through AWS reducing costs on less common operations like initial login.
3. **Storage Management** As mentioned previously, the recommended storage plan will utilize local and cloud solutions. For local storage an on-premises server can store all the reference images for the game as well as other reference material for the application. This will allow for cheaper storage of the relatively small files. The recommendation for saving game results and session data is to utilize AWS RDS for MySQL. Using a relational database will allow the class objects to be stored as records in a MySQL database. This database will provide fast access and allow for scalable size of the database over time. The last benefit of this recommendation is increased resilience of stored data as utilizing a multi-region plan will allow for backups in geographically separated datacenters.
4. **Memory Management**: The platforms recommend that most of the work of memory management is onto the systems themselves. The backend is written in Java which utilizes JVM. This means that the applications machine instructions are tailored to the operating system at runtime using just in time compilation. This allows the system to effectively manage memory the way the OS designers intended as well as provide built in garbage collection to keep memory free. The other main side is the client and by developing this to run in browser memory management is maintained by the browser since we are running our application in an application that handles memory management.
5. **Distributed Systems and Networks**: It is recommended that a client server model utilizing microservices be utilized. This will break each task of the system into an independent service that will use API calls to provide their specific service to the driving service. This architecture is perfect for distributed services as it is intended to communicate utilizing internet protocols, and ands a layer of robustness to the system. If one service goes down the rest of the services will continue to function, allowing for quick diagnoses and hopefully recovery to get full system functionality back. Additionally, by using AWS as the backbone for most server services the game will benefit from lower latency as traffic will be routed through AWS low latency network and global access points.
6. **Security**: Utilizing AWS for the storage of game records and web servers means that all user data will be in the cloud and not on the recommended on premises server which removes concerns about physical security as that will be AWS’ responsibility. As for virtual access, the two main concerns will be when data is at rest in the system and when it is being transferred from system to system. For in transit the APIs encrypt the data and transmit it utilizing best practices. For the system at rest, it will be protected by utilizing an authentication service to control who has access to the system. Also, by using an authentication service we will be able to use role-based access control which will ensure that data is on a need-to-know basis by having user accounts be created with the lest privilege principle.