

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 01/23/22 | Michael Ross | Added Executive Summary and Design Constraints content. |
| 1.1 | 2/6/22 | Michael Ross | Added Evaluation Content. |
| 1.2 | 2/18/22 | Michael Ross | Added Recommendation Content |

**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 Gaming Room is looking to provide a web version of their existing Android only game Draw It or Lose It. The web-based version of the application will allow it to reach a wider audience on multiple platforms.

## [Design Constraints](#_2et92p0)

* The software must utilize a singleton pattern to keep a single copy of the game in memory at a given time. This constraint means that to scale the application we will need to add additional servers rather than additional processes on a single server.
* Unique IDs will need to be generated for teams, games, and players to avoid duplicate entries. This constraint will require the use of iterators for searching and creating IDs and names.
* List objects will need to be utilized for Teams, Games, and Players to ensure that there can be more than one of each.

## [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)

Looking at the UML class diagram below it can be seen that the game, team, and player classes inherit from the Entity class. This is an example of inheritance. The GameService class acts as a singleton and is used to instantiate zero or more instances of the Game class, which can be used to instantiate zero or more Teams classes which in turn can be used to instantiate zero or more Player classes. The private nature of the properties of the GameService class is an example of encapsulation or abstraction. They can only be operated through use of public methods. The main class uses the testSingleton class directly to test the functionality of the GameService singleton.

The software requirements are fulfilled through the use of these classes in the following ways:

* The GameService class implements the singleton pattern through use of a private constructor. This directly fulfills a requirement that there only be one copy of the GameService in memory at a given time.
* The Game class includes a List object for teams which satisfies the requirement that there be more than one teams involved.
* The Team class includes a List object for players which satisfies the requirement that there be more than one player allowed per team.
* The GameService class implements methods to find the next Team and Player IDs to ensure that each player and team ID is unique since the GameService class is the only one running in memory due to the singleton pattern implemented. This satisfies the uniqueness requirement for teams and players.

**"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** | The Mac OS is based on the Unix OS and so supports many of the same software that Unix operating systems support.  **Advantages include:**   * Linux / Unix style OS supports many web server packages. * Low cost.   **Disadvantages include:**   * Mac OS is designed to be run on proprietary hardware so scaling can be a problem. * Most cloud instances that support MacOS are very expensive. | Linux based OS as the server side has many advantages:  **Advantages:**  -Most cloud providers offer Linux OS as an instance type. This makes scaling to thousands of players easier.  -Just about every server software supports Linux first and other OSes as an option. This increases choice in server runtimes.  -Linux servers have very low overhead when combined to other OSes. This makes them optimal for running server side binaries efficiently and performantly.  -Linux distributions can be free or very low cost. This can impact the total cost of running Linux for the server side.  **Disadvantages:**  Maintenance of Linux based OSes requires specialized personnel and skills. | Windows OS is used in many enterprise application servers because of the broad enterprise knowledge in Windows systems.  **Advantages:**  Many enterprise admins know and utilize Windows servers today.  **Disadvantages:**  Higher server overhead when compared to Linux and Unix OSes. This can impact effective scaling of the application.  Windows Operating Systems always cost much more in licensing compared to MacOS (<20$) and Linux (Free in some cases) | Mobile Devices are not often used for the server side because of many factors:  **Disadvantages:**  Mobile Devices are so varied that specialized Operating System accommodations need to be made for them. This specialization makes them not ideal for server side applications.  Performance is typically limited on mobile devices which will impact scaling capability of the server side. |
| **Client Side** | Since the development of the application will utilize HTML for all desktop applications, this negates a lot of the downside of developing for different Operating Systems. Mac will behave the same as Linux and the same as Windows as long as there is an HTML5 compatible browser. | Linux clients support running a graphical desktop environment like KDE, Gnome, etc. and also support running modern web browsers. Because of this the HTML version of the application will work the same on Linux as it does on Mac or Windows. | The HTML version that runs on Linux and Mac will also run on Windows just as well as Windows support all of the modern web browsers with Javascript and HTML5 support. This makes it much easier to develop an application which supports multiple OSes out of the box. | Running in a web browser on a mobile device can be achieved the same way it can on Windows, Mac, and Linux OSes. The question is whether it should be or not. Most users prefer the experience of a native mobile application from the relevant app store over the web experience on a mobile device. This is where the development path would diverge into different platforms which can take the single page application and make it into a native app, like Swift for Apple devices or Kotlin for Android. |
| **Development Tools** | Because we’re developing a mobile application using web servers and clients, any of the popular IDEs can be used for Mac. VSCode, Eclipse with Maven, and XCode are popular choices. These would all allow application development with syntax checking, app server testing, and app packaging. | For development tools on Linux, most of the same tools can be utilized that are run on MacOS. VSCode for code development, Eclipse, and the basic command line editor Vim can be utilized. For the server side, java is also necessary. | Developers running on Windows can use all of the tools listed for Mac and Linux. They can even utilize the Linux style command line using the Windows Subsystem for Linux (WSL). A lot of the server side tools are built for Linux like SSH SCP, and the like, but purpose built clones of these tools have been built for Windows with the GUI in mind like WinSCP and PuTTY. | The app stores feature prominently in the category of development tools for Mobile devices. There are testing services available for each. Apple has Testflight which tests all aspects of your application to ensure it is ready for the store. The Google Play store has Play Console for app testing.  XCode for Mac will need to be utilized to develop any Apple store code. |

## 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**: The recommended operating system platform for Draw It or Lose It is Linux running on a public cloud provider. From the ‘Server Side’ section of the operating system comparison table above it can be seen that Linux has many advantages and few drawbacks when it comes to running web-based server applications. All the major cloud providers support Linux in one of its flavors, and it has an incredibly small OS footprint so costs less than other OSes with larger resource footprints.

On the client side, the application will leverage native platforms where possible and web platforms where native platforms are not available or feasible. Leveraging native platforms on Apple, Android, and others ensures the user gets the experience they are accustomed to on their device. Leveraging a web client when this is not possible ensures that the application can run on any device using web as a fallback.

1. **Operating Systems Architectures**: The recommended architecture would be to utilize serverless wherever possible combined with a microservices based architecture. The reasons for this architecture are twofold:
   1. Serverless or Platform as a Service (PaaS) can be leveraged for application components like databases and persistent storage, which don’t do well in a microservice because of their stateful information.
   2. Microservices can be leveraged for stateless application components like web servers, which can run well in a container and which would access the PaaS services for state.

In addition to leveraging microservices and PaaS architectures, the front-end application will utilize APIs to communicate with the backend server infrastructure. This ensures that the application can run on any frontend that supports HTTPS communication to the backend server infrastructure.

Finally, we would recommend that the application uses a scale-out architecture as opposed to a scale-up. Because of the incredible forecasted demand for this game, scaling up a single server or even several servers to meet demand won’t be feasible. To fix this, the server-side application components will be built to leverage public cloud capabilities to scale out the web components when needed to meet customer demand.

1. **Storage Management**: Ideally for this application storage will be provided by the cloud provider in several different tiers:
   1. *Operating System Storage:* This is the storage needed to run the operating systems that the microservices use. These can be ephemeral and small as they are only needed to run the operating system itself.
   2. *Persistent Storage:* This would leverage services such as S3 or Blob storage and would be used to store larger amounts of data that need to be persisted and shared across all games.
   3. *Database storage:* This would be used to store information which needs to be accessed constantly and quickly such as game statistics and user account information. This type of data can leverage PaaS services provided by the public cloud provider such as DynamoDB in AWS and CosmosDB in Azure.

Leveraging the cloud provided storage allows for flexibility if the game wants to add more images for drawing or add additional players to the database as it allows addition of extra capacity without having to provision any extra hardware.

1. **Memory Management**: In general, system memory will only be utilized for information which is needed quickly and frequently. Examples of this in the application include:
   1. *Images for running games on the client side:* These images are very important to the game and the speed and accuracy at which these images render is critical to the game’s operation.
   2. *Client-Side Application Code:* The code of the application itself. This needs to be stored in memory to run.
   3. *Server-Side Application Code:* The code of the server itself. It is critical that the server side runs performantly as it will be coordinating with many clients per server.
2. **Distributed Systems and Networks**:

Communication will be handled in a few different ways depending on the components communicating:

* 1. *Microservice to Microservice:* Microservices will communicate with each other over local networks using APIs. Each microservice will have its own API which other microservices can use to communicate information to and from the service. This API acts as a contract between Microservices and ensures that each understand what the other is looking for in terms of authentication and parameters. This communication will leverage load balancers where possible to ensure uptime in the event of an outage.
  2. *Microservice to PaaS services:* Microservices will communicate with the PaaS services using the method applicable to that PaaS service over private or public networks. For example, a microservice would leverage RBAC and a connection string to communicate with a PaaS database service or just RBAC to communicate with a storage PaaS service.
  3. *Client to Server:* Clients will communicate with the server application using the server APIs. These APIs make it possible to communicate with the server side the same way from different clients.

1. **Security**:

Security will be top of mind for the application in general but will be applied at several key points in the application:

* 1. *Authentication:* Users will be required to create an account and login with a password. This will ensure only valid users can leverage game resources. Two factor authentication could be added but the complexity of adding such a system may not be warranted for the non-sensitive data being utilized by the game.
  2. *Authorization:* Users will be checked to ensure that they are the right user type to participate in a particular game.
  3. *Client to Server Communication:* HTTPS will be leveraged to ensure authentication and game information is transmitted securely from the client to the server.
  4. *Server Side:* The server side will leverage the principals of least privilege. Each microservice and PaaS service will only be provisioned with the minimum rights required to ensure that the service can perform its task properly.
  5. *Data Storage:* Data will be encrypted where possible to ensure exfiltration of data cannot be performed.
  6. *Logging and Monitoring:* Logging, monitoring, and alerting will be configured on the server side to ensure if there are performance, crash, or security issues the operations team responsible will be notified and corrective actions can be taken.