

# **Draw It or Lose It**

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**Draw It or Lose It** 1](#_Toc125121209)

[**CS 230 Project Software Design** 1](#_Toc125121210)

[Table of Contents 2](#_Toc125121211)

[Document Revision History 2](#_Toc125121212)

[Executive Summary 3](#_Toc125121213)

[Requirements 3](#_Toc125121214)

[Design Constraints 3](#_Toc125121215)

[Connectivity Constraints. 3](#_Toc125121216)

[UI Constraints 3](#_Toc125121217)

[Protocol Constraints 4](#_Toc125121218)

[Backend Constraints 4](#_Toc125121219)

[System Architecture View 4](#_Toc125121220)

[Domain Model 5](#_Toc125121221)

[Evaluation 6](#_Toc125121222)

[Recommendations 8](#_Toc125121223)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/20/2023 | Terry Sides | Initial Document Creation |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has a game application called, Draw It or Lose It. The game is currently written for the Android Device Operating System and deployed in the Android App Store. The company is interested in exposing the game to a wider audience by rewriting the application on a web-based platform and exposing the functionality to end users in a browser.

## Requirements

1. The game interface should be secured with at least a username and password.
   1. The username should be the users email address.
   2. The password should meet minimum security requirements.
      1. At least 8 characters in length
      2. One Uppercase letter
      3. One Number
      4. One Special Character
2. The game interface should include a page for new users to register with a username and password.
3. The interface should support the ability for a player to create a new game and invite new players.
4. The interface should support the ability for a player to join an in-progress game upon being invited by an existing game player.
5. The interface should support the ability to list the games that the current user is a player in.
6. The interface should support the ability for a player to drop out of a game.
7. The game interface will support the following browser implementations.
   1. Google Chrome
   2. Microsoft Edge
   3. Apple Safari
   4. Mozilla Firefox
8. The backend system will persist state data to keep track of games played, teams the user is a member of, and other relevant player data as defined by The Gaming Room.

## [Design Constraints](#_2et92p0)

### Connectivity Constraints.

The Draw it or lose it game is graphics intensive. Images will be sent over the internet at a relatively fast rate. The user’s ability to play this game effectively will be significantly impacted by the user’s connection to the internet.

### UI Constraints

This application is being developed in a web-based environment. It could be played from a desktop/laptop computer or from any mobile device with a web browser and a connection to the internet. The application will require device / browser sensing code that can determine the user’s platform. This information will then be used to direct the user to the appropriate view for that device and its display capabilities. It should be noted that for small handheld devices the images being displayed will likely be quite small.

### Protocol Constraints

HTTP is a stateless protocol. No state data is persisted from one request to another by default. This system will need to define a state mechanism for persisting data across web requests. This could be the URL, a backend database, the file system, the web server’s session or application object, or a combination of the above state management approaches.

### Backend Constraints

It should be anticipated that large numbers of players will be participating in large numbers of games at any given time. The system should be developed in such a way that this load can be distributed across multiple state persistent web servers. Web traffic should be load balanced across this server farm such that requests and responses are delivered to the correct server. This is known as sticky session load balancing.

Loading and storing large numbers of images on a web server is not recommended. The process is disk and memory intensive. For this reason, images should be served from a media server farm, that is itself load balanced. Where possible image caching should be used to reduce the disk read operations required to serve images.

## [System Architecture View](#_ilbxbyevv6b6)

Diagram

Description automatically generated

## [Domain Model](#_8h2ehzxfam4o)

The class diagram below shows the core gaming system structure as it pertains to the Game Service, Games, Teams, and Player relationships. Note that the Entity class is used as a superclass for the Game, Team, and Player classes. This is done to reduce the coding requirements and prevent duplication of code in derived classes. The Game Service is a singleton object implemented as a static class with a special method defined to obtain the instance. Only one instance per web server is allowed at a time. Care will be required to ensure that requests and responses are routed through to the server housing the correct game instance for that player and game. Note that the relationship defined between the Game Service and Games are zero to many. The same relationship exists between Games and Teams, and between Teams and Players.

It might be prudent to define maximums for the number of players per team, and the number of teams per game. Because the web architecture requires sticky sessions, not providing for some limits may overwhelm a particular server. In this case, the server will either respond very slowly, or crash all together. Servers are not unlimited resources in terms of memory and/or throughput. It may also be necessary to limit the number of games per game service. A smart routing system could be designed and layered on top of the application to re-route requests for new games to a server with a smaller load.

**"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)

1. Server Side
   1. Apple / Mac

While Apple/Mac hardware is certainly capable of handling a server-side workload, this is not what it’s primarily designed to do. Mac hardware is designed more for high end video/image processing, graphics, and CAD applications. Mac hardware is not manufactured for rack space applications. It would certainly be possible to install the Mac OS on commodity rack hardware, presuming one could locate the device drivers to make that installation fully functional, one would still incur the very steep OS licensing fees. Mac does support the NGINX+/Apache web server software.

* 1. Linux

Commodity hardware is widely available to run Linux server installations. Linux enjoys wide community support and open-source development. It runs quite well on rack mounted hardware and the licensing fees are minimal. Linux is a good option for deploying a web server farm using NGINX+/Apache.

* 1. Windows

Much like Linux, Windows enjoys wide industry support for it server versions of the operating system. Windows has a robust built in web server product called IIS (Internet information services). The windows operating system is well suited for rack space server farm type installations.

* 1. Mobile Devices

The nature of this application as it is currently envisioned would not involve using mobile devices for server-side operations.

1. Client Side
   1. Mac

The Safari browser application is certainly capable of running the client-side code that might be used in this application.

* 1. Linux

The Mozilla / Firefox browser application is capable of running the client-side code that might be used in this application.

* 1. Windows

Microsoft Edge can run the client-side code for this application.

* 1. Mobile Device

Any mobile device with a WC3 compliant browser should be capable of running this application.

1. Development Tools
   1. Mac – Eclipse / Java
   2. Linux – Eclipse / Java
   3. Windows – Eclipse / Java, Visual Studio / C#
   4. Mobile Devices – Eclipse / Java Mobile Workload

## Recommendations

1. **Operating Platform**:

Building the application as a web application and delivering it via Browser to the client allows for easily expanding the application across platforms and eliminates the need for overlapping development efforts across different hardware platforms like Apple and Android. In in this case, it would be our recommendation to move forward using Linux as the server platform.

Selecting the Linux Server Operating Systems provides a scalable server platform well suited to a high demand / high traffic web application. It is easily virtualized and well suited for migration to the cloud if the need arises. Linux has a mature support cycle, a long-term release program, a large open-source ecosystem, and enjoys a large industry pool of development resources. It’s multi-tasking, multi-threaded capability makes it a good choice for large volume web applications.

Linux is easily visualized using software Docker and Kubernetes. It supports robust LDAP (Lightweight Directory Access Protocol) user management. Linux also support industry standard web server software like NGINx and Apache.

1. **Operating Systems Architectures**:

The image above shows the basic architecture of the Linux platform. Running on modern multi-core processors Linux supports true multi-threaded / multi-tasking capability, it maintains process isolation, protects the kernel from external access, and provides efficient memory management / allocation.

1. **Storage Management**:

Both Linux and Windows support backend database products such as Microsoft SQL Server and Oracle. Either product would satisfy the requirements for this project in terms of backend application state storage as well as application runtime storage needs.

Storing 200 images that are 8 MB each in size will require about 1.6 GB of disk storage. This is a minimal amount of storage by today’s standards. A single terabyte drive could hold thousands of these images. The limiting factor in this case will be the speed at which the Operating System can read these images from disk.

In this case I would recommend a media server dedicated to caching and serving the images to the client from memory rather than individual reads directly from disk whenever an image is needed.

1. **Memory Management**:

Both Windows Server and Linux are multi-threaded / multi-tasking operating systems based upon a context switching architecture and modern multi-core processors. Both operating systems allocate protected memory space for processes at startup time. Both the Java Virtual Machine and the Microsoft Common Language Runtime offer garbage collection services. This helps off load memory management from the development staff and helps to protect against memory leaks. Both operating systems offer memory paging capabilities to swap memory to disk if the need arises thereby increasing the virtual memory available to the OS in memory allocation operations.

Once again, even if all 200 images are cached in memory that’s only 1.6GB of memory. If this task is off-loaded to a media server, the impact to the web server would be minimal.

1. **Distributed Systems and Networks**:

Developing the application in a web environment and delivery over HTTP accomplishes many of the distributed systems requirements by default. The entire work of rendering the User Interface is transferred to the client via HTML mark-up and browser rendering. The server platform is set behind a load balancing system and is implemented as a server farm with failover capability. A crashed or failed server is automatically taken out of the pool. In this case, images are served by separate media servers further spreading the workload out among several different machines. Further redundancy could be accomplished by migrating this entire system to the cloud (Azure or AWS) and building failover regions that could automatically redirect traffic upon a catastrophic server farm failure.

1. **Security:**

The first line of defense in the security space is HTTPS. This is a secure encrypted communications methodology implemented with certificates. It is recommended that the client make use of this technology.

The application will require login credentials from the user each time the application is accessed. As noted earlier these credentials will be subjected to minimum password requirements.

Data security will include at rest data encryption supported by both Microsoft SQL Server and Oracle. The application will make use of encrypted configuration files for storage of critical information on the web server such as connection strings and database credentials.

In this case a hardened firewall / load balancer such as F5’s WAAP product will offer state of the art threat detection, denial of service, and firewall services. The actual web servers are protected from direct client access.

It is recommended that if the Gaming Room wants to host this solution on premises that they implement some type of physical security to the facility.

1. **Development Considerations**

The Java development stack enjoys wide industry support and boasts are large community of professional developers. Using Linux, Java, and Eclipse as the development stack means the developers can work on the system locally and then push code to source control. Any development for mobile devices could also be done locally by the development staff on Linux workstations.