

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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)

[**System Architecture View 3**](#_Toc115077323)

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

**Evaluation 4**

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 1/22/23 | John Parsons | Initial release, executive summary and project overview. |
| 1.1 | 2/26/23 | John Parsons | Add Requirements, design constraints and recommendations, final draft |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room currently operates an Android based version of the game Draw It or Lose it. They are looking to expand their audience by also offering the game in a Web based environment. This will require porting the existing Android codebase to something more web friendly, as well as introducing cloud based infrastructure on the backend to satisfy game hosting requirements.

## Requirements

1. Game will allow for multiple teams to play together.
2. Game will allow for multiple players to be on the same team.
3. Game instances will be unique.

## [Design Constraints](#_2et92p0)

1. Game environment will be run in any modern browser (Chrome, Firefox, Edge).
2. Game environment will be written in a modern full stack framework.
3. Game will be written securely and with the utmost privacy concerns addressed.

## [System Architecture View](#_ilbxbyevv6b6)

TBA

## [Domain Model](#_8h2ehzxfam4o)

Each object in the application will be derived from a base class called Entity. All games, teams, and players will be created from this base class. A class called GameService will serve as the main factory behind accessing the individual classes for the different elements of the game. A game will contain one to many teams. A team will contain one to many players. Players, Teams, and Games will all have unique names and ids. A singleton topology will be implemented to ensure that every instance of the game that is created will be unique.

**"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**

**Server Side Web Server:**

Mac Server: Discontinued and never really became a standard in the computing industry. Not much more needs to be considered here.

Windows: IIS (Microsoft Web Server) is fairly popular with certain enterprise companies and can easily integrate with Azure for a cloud solution. Certain limitations are in place on what projects you can run on the backend (somewhat limited to Microsoft stack). There are much higher licensing prices as well when you go full Microsoft Stack. SQL Server can be very expensive as well.

Linux: The open source solution and in general has reduced licensing fees. Also the de facto standard for machine images when running in the cloud due to it being free software. Somewhat less turnkey than IIS but cloud providers have tried to make this more user friendly.

**Cloud Vendor:**

Deploying this backend solution in the cloud will be essential to ensuring that it is scalable and will offer the flexibility to pivot in direction based on user demand. It will save costs by allocating / deallocating resources based on how many users are logged into and using the system. The cloud providers are as follows:

AWS: Amazon web services, the most popular solution for cloud services on the internet. Depending on which tier you purchase you can have a suite of software solutions to help you maintain your cloud infrastructure. This includes automatic security patching, load balancers, and much more.

Azure: Microsoft’s offering, quickly gaining market share. Although their solution is not as robust as Amazons, they offer unique advantages like integrating directly with active directory. Also easily configurable into your SDLC if you are a Microsoft shop. In general this is a good solution if you only want to use Microsoft products.

Google Cloud: The third cloud provider, there is nothing extremely unique about Google other than they may be cheaper and possibly assist with search engine optimization. Their machine images and load balancers are about on par with Amazon.

**Client Side:**

In general there are a few solutions to providing a cross-platform experience:

1. React  
   If Draw it or Lose it is written in React it will run natively on web browsers on Android, iOS, and PC. React utilizes JavaScript which is supported on all modern browsers. Development time will be considerable as the android code would need to be ported to React.
2. Node.js  
   This is the cannon to kill the fly solution. Node.js is a huge library and offers everything under the sun to build your entire front end (and possibly backend) web application. This solution is a good pick if The Gaming Room plans to add new games or additional functionality down the road. Like with React,
3. JQuery Mobile  
   This is the legacy solution if you have developers who already know JQuery. Although it its not as robust as the other offerings, it will be quicker to spin up a solution and begin testing. The drawback is that it does not offer all the bells and whistles that more modern and contemporary frameworks offer. JQuery is old and may lose support in the next decade. That being said, a good majority of front end web pages still run on JQuery, so it is probably not going away any time soon. This solution will be the fastest to get up and running.

**Development Tools:**

**IDE**:

When using a pre-compiled language for the web application, an IDE is a necessity. There are many benefits to coding the backend in a pre-compiled language. One benefit is that debugging your code is much easier, as you can step through execution line by line in real time if one needs to. Another benefit is that IDEs are usually much better at spotting issues that will occur during compilation time. This is in comparison to run-time compiled languages where you may not see these errors until it is run.

One downside is that in order to do deployments the entire project must be compiled. You cannot simply make ad-hoc changes to the code without deploying the entire project. You lose flexibility in this way.

For IDE usage we have the following recommendations.

Any Microsoft stack language: Use Visual Studio, it is an amazing IDE and only continues to get better. The only downside is that it costs a lot of money, and you cannot run it on a non-windows machine.

Other Languages: Use a Jetbrains IDE. For Java and Python, Jetbrains is the industry standard.

**Text Editor**:

If the language is compiled at runtime then you will have more flexibility in your deployments in certain ways, although your debugging methods will have to change. There are a myriad of text editors one can use and many are compatible with multiple operating systems.

1. Notepad++:  
   This is an excellent editor that has support for plugins and syntax coloring for many different languages, always a good choice.
2. Vim (vi, gvim):  
   This is an open source editor that does have versions with a graphical user interface. Unless you were taught vi you probably want to avoid this one.
3. Visual Studio Code:  
   This text editor by Microsoft is compatible with Windows, Mac, and Linux. It has a huge library of plugins to run almost any language at some level. This editor is quickly gaining adoption as the standard editor to use on most platforms when working with runtime compiled languages. It also free.

**Addendum to Development Tools:**

Regardless of which editor one chooses, they must also consider how they will manage source control. Many IDEs have built in source control integration, this may be a deciding factor in which tool to use. There are different offerings of source control (TFS, cvs) but Git is by far the most widely used at this point. With that said, plugins and git integration should be considered when choosing development tools.

**Recommendation**

**Operating Platform**

The operating platform recommended for this implementation is a Linux flavor solution. In general, Linux is has a reduced price tag to implement and is easier to get up and running in a cloud environment. The flavor of Linux recommended would be Debian. The web server would be Apache, another mainstay. The actual backend code to handle Draw It or Lose It should be in .Net Core, a programming framework offered by Microsoft that is compatible with many operating systems. This will enable The Gaming Room more flexibility down the road if they decide to leverage a different operating system other than Linux (Microsoft based).

The Linux server and all its components will run in an AWS EC2 Instance, with more or less machine images to handle the load. The benefit of setting this up in AWS is that a lot of factors such as load balancing, security patching, storage, cpu management, memory management, and DDOS protection are built into the system, reducing overhead.

**Operating System Architectures:**

The operating system architectures are as follows:

The Debian Linux distribution will run on an Amazon EC2 instance. This machine image will house the Apache web server that hosts the public facing website. Linux will also run a MySQL server which will contain any necessary table information relevant to supporting Draw It or Lose it.

Apache will host the .Net Core application on the backend, which has libraries built int to access our MySQL database(s). The front end will be written in React, through asynchronous JavaScript calls the front end will communicate with the .Net Core application to retrieve any data necessary.

**Storage Management:**

The system will utilize Amazon’s storage management tools and configurations to properly handle hard disk space and usage. If necessary an individual or team at Amazon should be prompted with this document to further refine any additional storage configuration parameters. It is the findings of this team however that storage in its current iteration will not be an issue. Limits and warnings can be added to alert administration when certain thresholds are about to be surpassed.

**Memory Management:**

The system will utilize any tools that AWS has regarding memory management. Because .Net Core is a modern programming language, memory management and garbage collection are already built into the compiled application. Limits and warnings can be put in place when memory usage increases unexpectedly. Both memory and storage management can be implemented in an almost “fire and forget” approach, with the only drawback being that price increases may occur based on usage, hence warnings going out to administration upon preset usage limits.

**Distributed System and Networks**

From a user’s perspective there will be no noticeable difference between playing on a mobile device versus PC. Players from both platforms will communicate with the same systems and the calls will be identical. Players from both platforms will be able to play against each other. Any distribution of machine instances will be implemented in a disaster recovery (DR) like solution. The Gaming Room will work with Amazon to implement a fail-over server in a different region which is synchronized with the main server.

In the event that the main server were to become unavailable, EC2 would be configured on the backend to immediately switch all incoming connections over to the DR zone. Game state should be captured and transferred to the other site almost instantaneously via log shipping. In the event of a failure mid-game, reconnection to the DR zone should take place within 60 seconds. Any new user would have a seamless experience as they will unknowingly be playing on DR.

**Security**

In general there will be two layers of security, one leveraging any available tools from Amazon (automatic OS patching, DDOS protection, etc) and the other being built into the code itself (authorization/authentication, parameter whitelisting, privilege escalation, other OWASP top 10 items). AWS has a suite of tools to help lock down your system and ensure it is as secure as possible. It is recommended that either this team or someone at The Gaming Room reach out to Amazon to understand all offerings available.

The other level of security is more custom in that it is within the game code itself. The program will be written to ensure that users can only utilize the system once they are properly authorized and authenticated, using a least privilege mindset. Any parameters sent to the system will be checked against a whitelist for name and type. When accessing data, checks will be in place that will confirm the user is authorized to view the data.

Protections against cross site scripting and SQL injection will be implemented on the client side. The program will be written in both the front end and back end with security being the number one priority. A package manager should be utilized (usually through VSCode or Visual Studio) to update client side libraries.