

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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)4-5

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/17/20 | Janera Dobson | The Gaming room would like to develop a web-based game called Draw It or Lose it which is currently available as an Android app only. |

**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 currently has an application called Draw It or Lose It, only found on an Android OS. They have come to the decision that they now would like their app to serve multiple platforms for a web-based application that mirrors the functionality of their current app but are unsure of how to go about it. Specifically, the client has requested that the following software requirements be met for the game application:

* 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 to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* We must consider security, performance, user interface/user experience, knowledge of the framework, and platforms along with scalability.
* The need for two different hardware and software environments to suffice the necessity of moving from an app on a single client and OS platform to multiple web-based distributed environments.
* There is the need to pay keen attention to various aspects such as screen resolutions, sizes, interactions, and pixel densities for the web-based platforms to run fluidly across as many devices with as many screen sizes as possible.
* When considering any OS, the UI objects and patterns must be studied to develop and run the application without crashes or bugs ensuring the design performs well on all available web-based platforms as it does on the Android mobile app.
* Seeing there will be multiple players, connectivity is key but if the players are geographically dispersed and far away from the datacentre without the proper bandwidth, the higher the latency.
* Browser compatibility raises issues because different browsers come with different specifications and formats.

## [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)

The UML class diagram provided below is represented by aggregation meaning the structure of the diagram shows that there is a cluster of components that all relate to one another. Game, Team, and Player all have a "is a" relationship with Entity which means that all three receives components of Entity. Game, Team, and Player all have the same id and name attributes in common. There is a relation between Team and Player which would be a "has a" relationship. This means these classes have a reference to part of another class or of the same class with one another. Likewise, Game uses some reference to Teams and GameService uses some reference to Game. GameService has a reference of Game, Game has a reference of Team, and Team has a reference of Player. Therefore, each class relies on the infrastructure of another to gain complete functionality of the overall production of the game itself; giving rise to one GameService may have multiple games in turn each game can have multiple teams just like Team may have multiple players.

****

## [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** | -Free of charge.  -Easy to configure and implement right from the built-in terminal.  -Supports SSH for secure remote administration from the command line  -Must be kept secure from hackers and crackers.  -Must maintain a server farm for users to store data in the cloud.  -Latency issues.  -NodeJS | -Free of charge.  -Popular when it comes to servers due to versatility.  -Easy to configure and implement right from the built-in terminal.  -Must be kept secure from hackers and crackers.  -SSH (Secure Socket Shell)  -Must maintain a server farm for users to store data in the cloud.  -Latency issues. | -Beginner-friendly with GUI.  -Terminal must be installed and configured.  -Drivers for up-to-date hardware are easily available.  -Supports large numbers of third parties.  -Easy and optional automatized system updates.  -Vulnerable to malware.  -License cost per user.  -integrates "out of the box" with Active Directory-based corporate servers. | -Security risk  -would have to couple the client to server  -HTTP servers  -allows for hosting personal web applications  -servlet containers such as Jetty, Tomcat, Glassfish, and Resin.  -Configuration allows for optimization, account management, and synchronized information across platforms. |
| **Client Side** | -Medium level of expertise necessary.  -Time-consuming.  -Minimal load is required so specifications aren't substantial.  -JSON for communication.  -Costly and requires license service fees. | -High level of expertise necessary.  -Time-consuming  -Minimal load is required so specifications aren't substantial.  -JSON for communication.  -Minimum license cost. | -Minimum level of expertise necessary.  -Time-consuming.  -Minimal load is required so specifications aren't substantial.  -JSON for communication.  -Costly and requires support license fees. | -Native Apps, stored, or run on locally on a device.  -Mobile Web Apps, runs inside or is accessed via a URL browser on a mobile device.  - JSON for communication.  -Flexible to clients and developers.  -Challenging to implement. |
| **Development Tools** | -Swift  -C/C++  -Objective-C  -Ruby  -SQL  -Java  -PHP  -HTML  -CSS  -JavaScript  -Homebrew  -Xcode  -Eclipse  -Github  -iTerm2  -Visual Studio Code  -Intellij IDEA  -PyCharm | -C/C++  -Java  -Python  -JavaScript  -HTML  -CSS  -Github  -Shell  -Sublime  -Atom  -Github  -Brackets  -Eclipse  -KATE PyCharm | -JavaScript  -Python  -Java  -C++  -C#  -CSS  -PHP  -Perl  -HTML  -Microsoft Visual Studio  -Netbeans  -PyCharm  -Eclipse | -Eclipse  -Java  -HTML  -JavaScript  -CSS  -Apache Cordova  -Android Studio  -Xcode  -Windows App Studio |

## 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 operating platform I recommend which will allow The Gaming Room to expand Draw It or Lose It to other computing environments is ASP.net. This platform is impeccable for building desktop and mobile applications. You can build large apps with less coding, has a strong environment where memory leaks are easily spotted and neutralized, its fast and scalable, and most importantly, is supported on macOS, Windows, and Linux.
2. **Operating Systems Architectures**: ASP.net's basic architecture is Language, Library, and CLR (Common Language Runtime). ASP.net has a wide variety of languages in its framework which includes standard class libraries and has a CLR that deals with exceptions and garbage collection.
3. **Storage Management**: ASP.net uses a stateless protocol which manages four types of states: view state (page and all its controls), control state (can't be modified), session state (when the user accesses the website, a session is created), and application state (collections of all web page data).
4. **Memory Management**: ASP.net would use garbage collector memory management techniques for the Draw It or Lose It software. It would allocate and release only the parts of the game as needed and what does not will be destroyed by the garbage collector. For instance, if an object in a game is being referenced then it becomes available to the application, but once the object is no longer reachable by the application, the garbage collector will remove it and reclaims the unused memory.
5. **Distributed Systems and Networks**: Knowing that the client would like Draw It or Lose It to communicate between various platforms, this may be accomplished with the System.Web.Services Namespace. This namespace allows for multiple users to be in different areas to communicate and exchange messages using different domains. Because the application will be created for users anywhere it is best to use this structure for simplicity and interoperating with other users of the application.
6. **Security**: Security is a must-have for the client. ASP.net Cor Security has a way to protect user-information on and between various platforms by providing tools and libraries to secure apps built right in. They have authentication and authorization features, along with data protection and HTTP enforcement. Therefore, users will authenticate themselves by providing credentials to gain authorization.

Reference:

ASP.NET - Managing State. (n.d.). Retrieved October 16, 2020, from https://www.tutorialspoint.com/asp.net/asp.net\_managing\_state.htm.

ASP.NET: Open-source web framework for .NET. (n.d.). Retrieved October 16, 2020, from https://dotnet.microsoft.com/apps/aspnet.

Gund, J. (n.d.). .NET Memory Management. Retrieved October 16, 2020, from https://www.c-sharpcorner.com/UploadFile/26b237/net-memory-management/.

Pearson IT Certification. (n.d.). Retrieved October 16, 2020, from https://www.pearsonitcertification.com/articles/article.aspx?p=31490.

Rick-Anderson. (n.d.). Overview of ASP.NET Core Security. Retrieved October 16, 2020, from https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-3.1.

What is ASP.NET? and it's ARCHITECTURE. (n.d.). Retrieved October 16, 2020, from https://www.guru99.com/what-is-asp-dot-net.html.