

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)3

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

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/15/2020 | Mitchel Barton | Adds support for player and team classes; incorporate base class to hold common attributes. |

## [Executive Summary](#_sbfa50wo7nsh)

The client, The Gaming Room, is seeking to adapt their Android mobile game, *Draw It or Lose It*, to a web-based application that serves multiple platforms. The application will be first developed in Java, to later be adapted to a JavaScript format for inter-compatibility. The client should be aware that additional time may be required to transition from their current Android-only framework to the JavaScript format, if necessary.

## [Design Constraints](#_2et92p0)

The application needs to web-based and compatible with multiple platforms. It must also be able to support one or more teams involved, with each time having multiple players assigned to it. The game and team names need to be unique, and only one instance of the game should exist in memory at any given time.

## [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 Game, Team, and Player classes all have an “is a” relationship with the entity class. That means Game is an Entity, Team is an Entity and Player is an Entity. Game, Team and Player all are inherited from Entity. In UML it can be represented by inheritance. Game, Team and Entity, all 3 have common attributes as Id and Name. Therefore, it is defined at Super class Entity.

The relationship between the Team and Player classes is of the “has a” variety. A Team has Players, like a Game having Teams and a GameService organizing (having) Games. In UML this is represented by aggregation. A “has a” relationship means that an instance of one class has a reference to an instance of another class. In the UML diagram below, GameService has a reference of Games, while Games has a reference of Team and Team has a reference of Player. One GameService can have multiple games, each Game can have multiple Teams, and Teams comprise of multiple Players.

****

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Flexible terminal commands to configure the server, access, or make changes. The disadvantage is it runs on a limited hardware set, limiting options on that front. | Linux systems are renowned for their stability and reliability, while being cost effective. It does require more technical knowledge to run than the other platforms. | Windows has one of the widest ranges of software support compared to other large platforms. The cost of operation is like MacOS. | The server side is privy to a wide range of customization options. It is better if the server is immobile and tracked from a single place. The hardware specifications are better in other devices. |
| **Client Side** | Moderate expertise is required, with cost being similar to Windows. The hardware from Apple is more expensive than Windows machines | The cost is extremely low, but the technical expertise is much higher than other platforms; more time must be devoted to learning navigation and operation of the system. | Minimum expertise and time required; the cost is similar to MacOS. High hardware compatibility. | While slightly more difficult to implement than other ecosystems, mobile provides flexibility to the developer or client to see updates at any time. |
| **Development Tools** | Objective-C is the language most commonly used with MacOs. | Linux has able support for many programming languages, including C, C++, Java, JavaScript, Ruby, and Vala. | C++ is the workhorse language at Microsoft, being a static, free-form, multiparadigm, general purpose language. | For building iOS applications, Swift is the preferred language due to its’ ease of use and compactness. Swift is free and open source under the Apache 2.0 open source license.  For Android, the object-oriented Java is still the preferred method to write native applications. |

## 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 Gaming Room requires that *Draw It or Lose It* be on a web-based platform, so that it can be compatible with all the major operating systems. Since the client already has experience developing in a mobile environment, that is the platform the web-based version can be developed on.
2. **Operating Systems Architectures**: A mobile application typically includes two components: the mobile platform and an Infobase. The Infobase contains an equivalent of a file database (for storing user data) and a mobile application (programming code executed on the device).
3. **Storage Management**: The primary server-side framework should be written in Ruby on Rails. It is a model-view-controller framework, providing default structures for a database, web service, and web pages while being extremely dry to code in.
4. **Memory Management**: The application will be web-based, so memory management should be left to the JavaScipt (JS) implementation. JS utilizes a form of automatic memory management known as garbage collection (GC). The purpose of a garbage collector is to monitor memory allocation and determine when a specific block of memory is no longer needed and reclaim it. However, automatic garbage collection (like what is present in JavaScript) does not completely free us from worrying about memory management on large scale apps (Ayesh 2018). Some other steps to be taken during development should include the avoidance of unbounded objects, use of primitive values, and deserializing files directly if they are being handled server side.
5. **Distributed Systems and Networks**: Since The Game Room is aiming to expand their reach to multiple platforms scaling the database of the application is necessary to handle the increased query load. At the beginning this can be accomplished through the Primary-Replica Replication strategy, where two new database servers are synched to the primary one. As the project gains more traction, that database should be migrated to a sharded system. Here, the server is split into multiple smaller servers called shards. These shards all contain different records, which need to be spread uniformly during implementation. It is not an easy technique to implement, but it will give the client to increase write traffic to *N* times- where *N* is the number of shards.
6. **Security**: The best practice for securing data is to protect data at rest and in transit. The use of encryption, like a binary search tree, can used to protect data at rest, while the use of a high-speed WAN link like ExpressRoute should be used to secure data sets during transit.

Works Cited:

Ayesh, Kareem. Sept. 14, 2018. “Large Scale Web Apps Memory Management”. https://medium.com/@kareemayesh/large-scale-web-apps-memory-management-b6dc06237836

Kozlovski, Stanislav. April 27, 2018. “A Thorough Introduction to Distributed Systems”. <https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c/>

Mozilla Developer Network. March 4, 2020. “Memory Management”. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory\_Management