

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0.0

## 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)5

[**Recommendations**](#_m8aleynsvzvc)8

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0.0 | 06/13/22 | Jad Alrehaoui | Final Design Document |

**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)

Draw it or lose it is a game where teams compete to guess what is being drawn. The application will render images from a large library of stock drawings as clues. The game consists of 4 rounds of play, each 1 minute. Drawings are rendered at a steady rate and complete on the 30 second mark. If the team does not guess the puzzle in time, the remaining teams have an opportunity to offer 1 guess each and solve the puzzle in 15 seconds.

## [Design Constraints](#_2et92p0)

**Technical Constraints :**

* Framework in use: Java
* Recommended OS: Linux
* Environment Hosting: Game Room
* Hardware Requirement: A high specs server to be able to handle multithreading and HTTP request for a big amount of clients
* Connection: Ping must not go over 50 ms so that we don’t experience bad UX and lag

**Business Constraints :**

* Development period: 3 weeks after confirmation
* Testing period: 1 week after finalizing the development phase
* Team Lead and Project Manager : Jad Alrehaoui

## [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)

* GameService (Singleton): This class will hold the only one instance of the game running. All games are found in a List of Games inside this class. We can refer to this class as a Singleton and we will implement the Singleton Design Pattern for this class. This class will give us access to the game we need to query, the nextTeam, nextPlayer, and how many games we are hosting.
* Game: This class will hold the information of the game present in the GameService Game List. It consists of the ability to add a team to the game, and a list of teams competing in the game.
* Team: This class will hold a list of Players, consists of an ID, and the ability to add player to the team.
* Player: This class will hold the information about a player, consists of an ID and a name.
* Entity: This class holds the common attributes between Player, Game, and Team. It consists of an ID and a Name.

**Relationships:**

1. An Entity **is a** Game, Team or Player.
2. A game can **have Zero or many** Teams.
3. A team can **have Zero or many** Players.
4. A GameService **has Zero or many** Games.

**GameRoom Initial UML**

**Diagram

Description automatically generated**

**How it works**

A GameService will host all the games on the platform, each game will have a unique name and a unique incrementing ID. These unique names and IDs are only unique inside the Game Entity. The starting ID for the first game created will be **1**.

A Game which is an Entity, will contain a List of teams, and will have the ability to create a Team. The team created will be available in the game created and in the teams provided by the GameService. A team name will be unique as well as its ID. The team name **can be** the same as the game’s name since it’s a different entity.

The Team which is also an Entity, will contain a list of players, and will have the ability to create a Player. The player created will be available in the team created and in the players provided by the GameService. A player **can have** a Team’s name, or a Game’s name as well as the ID. All Entities IDs start from one and incrementing.

Note: A different solution in which all names across all the entities are unique will be provided as code. If this is the requirements we will need to switch to that version.

## [Evaluation](#_Evaluation)

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** | **Pros**   * Less Virus Attacks * Good Support * Performance   **Cons**   * Expensive * No hardware customization * Less hardware used | **Pros**   * Open source * Secure * Fast * Low System Specs   **Cons**   * No Support * No standard edition * Difficult to troubleshoot | **Pros**   * Beginner friendly * Support large number of third party * Long term Support   **Cons**   * High licensing cost * Security related errors * Virus attacks | **Not applicable** |
| **Client Side** | Web-based applications work on MacOS through the web browser.  We can build a web-application to run in the browser and all Operating System can run it through the web browser | Web-based applications work on Linux through the web browser.  We can build a web-application to run in the browser and all Operating System can run it through the web browser | Web-based applications work on Windows through the web browser.  We can build a web-application to run in the browser and all Operating System can run it through the web browser | Web-based applications work on Mobile devices through the web browser.  We can build a web-application to run in the browser and all Operating System can run it through the web browser |
| **Development Tools** | Java can run on all operating Systems.  A JDK is required.  IntelliJ IDEA on MacOS is our preferred tool to develop this application.  Development on MacOS is smoother  Because it manages memory better than windows. | Linux can run Java.  A JDK is required.  IntelliJ IDEA works on Linux | Windows can run Java. A JDK is required. Eclipse is the preferred IDE for Java on Windows.  IntelliJ IDEA works on Windows | No development is made on Mobile Devices.  Swift with Xcode on iOS  Java with Android Studio on Android |

**Server**

**Why we need a server and what are we going to deploy on it ?**

Our software design consists of a client-server architecture. The server will act as a provider of APIs where all our logic and communication with the database will be implemented. We need a server that we can trust to give access to clients and perform function to meet their requests. We will develop a Java application that will listen and respond to requests. Java works everywhere, so there is no constraints to choosing a server. We will be using IntelliJ IDEA to develop this application that also comes at a cost of $49.90 US/Month/User.

**Mac OS Server:** There are two types of licensed, 10-client license and unlimited license, each has a price tag and feature.

1. The **10-client license** does not offer file sharing. This can be used to host a web server. The actual cost of this license is $499 US, and you have the ability to upgrade at any time to the unlimited version.
2. The **unlimited license** offers file sharing and can be used as a web server. The cost of this license if $999 US.

Apple comes up with a new version every 12-18 months. Every time there’s an update you will have to buy the new license for the newer version, or you can buy something called Apple Maintenance Program and keep having updates for 36 months. Another alternative is that you buy the current version license and do not update your server.

An alternative solution is using Mac OS Client, which is based on the code base, UNIX. The only difference between the client and the server is that the server comes with administration tools.

Reference: [macgasm.net](https://www.macgasm.net/news/tips/os-server/#:~:text=For%20OS%20X%20Server%20the,be%20done%20at%20any%20time.)

**Linux Server:** In order to deploy a Linux Server, we will be dependent on a variant of Linux. There are many types of variants, but the most common ones are:

1. **Red Hat Enterprise Linux Workstation**:
2. Self-Support for 1 year at $179 US
3. Standard for 1 year at $299 US
4. **Red Hat Enterprise Developer Linux Suite:**  self-support for 1 year at $99 US
5. **Red Hat Enterprise Developer Workstation:**
6. Professional for 1 year at $299 US
7. Enterprise for 1 year at $499 US

**Many more …**

Reference: [linuxhint.com](https://linuxhint.com/redhat_linux_pricing/)

**Microsoft Windows Server:** 5 reasons why windows server stands out

1. Easy to use management interface on both standard edition and datacenter edition.
2. Support for Windows server software such as IIS
3. Supports higher end hardware than standard Windows licensing
4. Faster software setup time on GUI interface
5. More hands on support available for novice users

**Pricing Options**

1. Standard Edition at $20 US/Month or own at $972 US
2. Datacenter Edition at $125 US/Month or own at $6,155 US

Reference: [servermania.com](https://www.servermania.com/kb/articles/how-much-does-a-windows-server-cost/)

**Client**

In order to reach as many clients as possible, a web interface should be built so that each operating system such as Mac OS, Linux and Windows can be reached through their built-in or other web browser, and a mobile application that acts as a client should be available on iOS for iPhones and Android OS for androids.

**Web Interface:**

1. **HTML/CSS/Javascript** – A web application can be built, designed and implemented using these three languages.
2. **ReactJS** – is a javascript library that interacts with the web browser’s DOM and create components with their functionality and styles in order to work with the server.
3. **AngularJS** – is also a javascript library that interacts with web browser’s DOM.

**Tools needed:** a text editor or an IDE, like Atom, Visual Studio Code. They are free to use.

**Brief:**

HTML/CSS/Javascript are the basics of web client programming so developing a client using them without relying on a library will be time consuming and some features may be impossible.

ReactJS is well known, and many companies rely on it to be their number one client side library.

It can make development easy and less time consuming. It has many available packages that can help developers only focus on the functionality and design.

AngularJS is also well known and developed by Google. It’s reliable and widespread. Many companies rely on this library to be their main client side because it has many packages that help developers reach their goals and designs.

**Mobile Interface:**

* 1. **Native tools:** Such as Swift for iOS and Java for Android
  2. **Cross Platform Technologies:** Such as React Native, Angular and Flutter

**Tools needed:**

For Native tools XCode for macOS and Android Studio for Android.

For Cross Platform Technologies: any text editor, like Atom, Visual Studio Code or IDE.

**Brief:**

Native tools will guide us to hire more employees to achieve the same functionality on both operating systems. They integrate well with the operating system, and they work closely with the mobile hardware. These tools will be time consuming since two teams will be working to build two applications that work the same way on different platforms. Also, the two application will not look exactly the same on both platforms since some built in widgets for each platform behave and look differently.

Cross Platform Technologies will allow us to work once on the client application and have it work on both operating systems. Some are better than other performance-wise. This will not take too much time designing and implementing the client application since we will be working on only one application.

**Publishing:** For both operating systems there are some fees when we need to publish the application

**iOS:** $99 US/Year

**Android:** $25 US Once

## 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**: We recommend Linux, it is flexible and easy to maintain. 70% of big applications are deployed on a Linux server disregarding the variant. It is good for expanding and adding more functionality, it only requires more configuration but less money.
2. **Operating Systems Architectures**: We recommend going with the Client-Server architecture since the application consists of a centralized data center that will hold everyone’s information and scores and is going to take decisions on runtime.
3. **Storage Management**: We recommend the storage to be a cloud-based storage since it has higher availability and management, the storage and the server will be in the same virtual private cloud VPC, and we can use AWS for that matter and AWS offers us a way to configure our storage to automatically scaled out or in based on the remaining space on our instance.
4. **Memory Management**: We recommend coding taking into consideration that we want to be as efficient as possible, no need use assets that are needed in objects, pictures and assets must be compressed to an acceptable kind of presentation to make the application smoother and faster on the UX side.
5. **Distributed Systems and Networks**: We recommend AWS services, if well configured, AWS handles the project’s scalability and maintenance in a really good way. Our project will consist of our application deployed on a Linux EC2 instance. The instance will be communicating with RDB – Relational Database Service – where all our user info and teams are stored. Also, the EC2 instance will have a connection to a S3 bucket which will be the storage of our application.

The EC2 instances will listen to requests coming from the ELB – Elastic Load Balancer – which will redirect all requests coming from all the user to our servers equally which makes the CPU usage of each server moderate.

1. **Security**:
   1. An TLS certificate must be used on the server and the client to have the data encrypted from end-to-end.
   2. Spreading the requests upon multiple servers will also lower the risk of a downtime which is also a security concern.
   3. We will only share what is needed to be shared, nothing, respecting the least privilege principal but on a data level.
   4. Since our servers are on the cloud no need for physical security to be established, or to isolate the server in a separate secured room.
   5. We will use IAM users to authenticate with AWS and to configure the cloud, as well we are going to restrict everyone access to only what is needed.
   6. EC2 instances and services will have policies attached to them so they can’t go out of scope.
   7. We will make use of security groups available on the AWS Cloud and we will leverage them to the maximum.