

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 | 05/20/2020 | Winston Spencer | Initial Draft |

**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)

Our client “The Gaming Room” has reimagined the 1980’s hit television game Win, Lose or Draw. They have developed a mobile version of the game called Draw It or Lose solely for the android platform. The client is now seeking to expand the game. This new version of Draw It or Lose will allow users across multiple platforms to compete via the internet.

## [Design Constraints](#_2et92p0)

1. **Game Platforms**
   1. The game is web based. Regardless of where is game client lives (IOS, Android, Browser), the game client will require an internet connection. An internet connection is required to allow users across multiply platforms (IOS, Android, Web Browser) to compete.
2. **Programming Language & Tech Stack**
   1. The services will be written in java. Java is the most popular programming language in the world. This allows us to onboard software engineers quickly.
   2. The services that support the game client will consume RESTful APIs. RESTful API’s offers great performance, security, flexibility, and speed over the internet. Frameworks such as spring boot, drop wizard allows engineers to quickly implement secured functionality. HTTP Methods also allows CRUD like operations such as:

* Create Game (HTTP Method POST)
* Retrieve Game (HTTP Method GET)
* Update Game (HTTP Method PUT)
* Delete Game (HTTP Method DELETE)
  1. The services will use the spring boot framework. As mentioned above, spring boot allows software engineers to build RESTful services in minutes. Spring boot integrates metrics (via spring-boot actuator), database (via spring-data-jdbc) and security (via spring-security).

1. **Versioning & CICD Pipeline**
   1. The application code will be stored in a code repository such as GITHUB. This allows the software engineers to build the application in increments/versions. Engineers can also revert to older versions of the code if needed.
   2. An automated continuous integration and delivery (CICD) pipeline should be created to build, test, and deploy the application whenever there is a change to the code. This allows the engineers to automate the build, test, and delivery process.
2. **Security**
   1. The application will use an Identity Management solution (such as PingFed) to manage authentication. This gives users the confidence that their game will not be manipulated by other users.
3. **Hardware Infrastructure**
   1. All RESTful services should be deployed in the cloud to reduce cost and easily manage scalability.
   2. The application should store its data in a scalable distributed relation database management system such as Azure SQL or Amazon Aurora.
   3. The application RESTful services should consolidate and store all logs in Splunk.
   4. Prometheus & Grafana should be used collect and report all metrics.
   5. PagerDuty should be used to notify the team of all system failures/outages.

## [System Architecture View](#_ilbxbyevv6b6)

Diagram

Description automatically generated

## 

## [Domain Model](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>



In the UML diagram above, the Game, Team and Player class all extends the Entity class. This concept is called polymorphism. This simply means that the Game, Team and Player class are also Entity objects. They will inherit the Entity class attributes and members. The GameService class utilizes the singleton pattern. At any given time, there will only be one instance of the GameService object in memory. The GameService object will manage zero or more Game, Team and Player object.

## [Evaluation](#_2o15spng8stw)

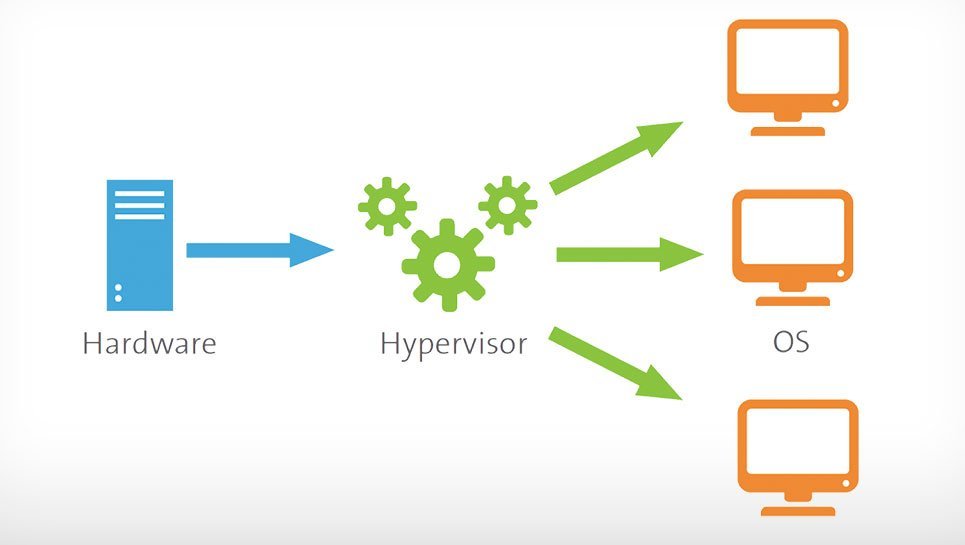
Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | PROS:   * Unix based OS ideal for hosting web-based applications * Strong Graphics capabilities for GPU computing * Great Management UI * Supports Kubernetes * Lower power consumption with the new A1 chip   CONS:   * Not a lot of customizable options | PROS:   * Economical (Some distributions are free.) * Unix based OS ideal for hosting web-based applications * Many customizable options * Supports Kubernetes * Scales easily   CONS:   * High Power Consumption * Complex Graphical UI * Requires command line knowledge | PROS:   * Windows OS ideal for hosting web-based applications * Great Management UI * Many customizable options * Supports WSL (Windows Subsystem for Linux) * Supports Kubernetes * Scales easily   CONS:   * High Power Consumption * Requires Hardware + License * Requires command line knowledge | PROS:   * Low Power Consumption * Economical   CONS:   * Limited to developers with Mobile Development knowledge] * Unix based OS NOT ideal for hosting web-based applications * Not a lot of customizable options * No Fiber Networking Option * Does not support Kubernetes * Does not scale easily |
| **Client Side** | PROS:   * Supports Java * Supports C# & .Net * Great intuitive UI   CONS:   * Premium Hardware Cost * Native development may require ObjectiveC expertise | PROS:   * Economical * Supports Raspberry Pi * Supports Java * Supports C# & .Net   CONS:   * Requires Linux expertise * Native development may require C++ expertise | PROS:   * Economical * Supports Raspberry Pi * Supports Java * Supports C# & .Net * Great UI * Very Familiar UI   CONS:   * Development may require .NET expertise | PROS:   * Economical * Cost may be subsidized by Mobile Carrier. * High User Adoption Rates * Supports Java/Kotlin for Android OS * Supports ObjectiveC for IOS * Very Intuitive UI   CONS:   * Native GUI Development may require Android or IOS expertise |
| **Development Tools** | Programming Language:   * Objective C * C++ * Python * Kotlin * Java   IDE:   * XCode * Eclipse * NetBeans * Visual Studio Code * IntelliJ | Programming Language:   * C++ * Python * Kotlin * Java * C#   IDE:   * Eclipse * NetBeans * Visual Studio Code * IntelliJ | Programming Language:   * C++ * Python * Kotlin * Java * C#   IDE:   * Visual Studio Code * Eclipse * NetBeans * IntelliJ | Programming Language:   * Objective * Kotlin * Java   IDE:   * XCode * Visual Studio Code * IntelliJ |

## 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**:
   1. A Linux platform on the cloud is recommended for server-side processing. Linux OS does require some technical expertise. However, new cloud-based services provided by Amazon AWS and Microsoft Azure can help mitigate this issue. The entry costs for a cloud-based Linux platforms can be free and scaled as needed. As the system is scaled up, the cost will increase.
2. **Operating Systems Architectures**:
   1. A 64-Bit cloud-based OS is recommended. 64-Bit servers support significantly more CPU cores, Memory and Storage capacity (<https://access.redhat.com/articles/rhel-limits>). The increase capacity allows multiple companies/clients to share hardware cost. This concept is at the center of cloud computing. The cloud consists of multiple servers. On top of each server hardware is a virtual machine monitor (VMM) also known as Hypervisor. Each hypervisor can support multiple operating systems. Because Linux is easily scalable, the client’s can have one or more Linux OS living on one or more hypervisors for maximum redundancy redundancy.



1. **Storage Management**:
   1. A cloud-based storage is recommended. This is extremely cheap and simplyfied. Cloud-based storage can support multiple file systems such as ETX-4, XFS, GFS or NTFS. This is abstracted from the end user. Cloud storage providers are increasingly using Non-Volatile Memory express (NVMe). “The NVMe architecture brings a new high performance queuing mechanism that supports 65,535 I/O queues each with 65,535 commands (referred to as queue depth, or number of outstanding commands).” (NVM Express, Inc., 2021) As mentioned above, this allows the customer to share resource cost.

Diagram

Description automatically generated

1. **Memory Management**:
   1. Cloud-based operating systems simplify memory management. Memory can be scaled up or down per host based on the customer needs. A 64-Bit operating system can support significantly more memory than a 32-Bit operating system. This means the customer can run more processes on a 64-Bit server.
2. **Distributed Systems and Networks**:
   1. Cloud-based platforms are inherently distributed. Multiple virtual machines can live on one or more hypervisors across one or more datacenters. Each server can be clustered on the same virtual private network. If one server, hypervisor or datacenter fails, the client application should not be affected. The traffic should simply route all traffic to another server on a hypervisor in another datacenter. It is also recommended that the client utilizes a cloud-base database server such as Amazon’s Aurora or Microsoft AzureSQL. These databases are inherently redundant and the complexity is abstracted from the end user.
3. **Security**:
   1. Security is always the cornerstone of the cloud-based architecture. Cloud providers ensure that multiple companies can share a hypervisor securely. That’s because security is built into the hypervisors from the ground up. The client can be assured that their cloud-based virtual machines are secured on their own private network and only accessible by them. We are also recommending an Identity Management services such as Ping Federation for the game client. This will ensure that each end user is connecting to their game alone.